DOCUMENT RESUME

ED 036 931	40	EC 004 963

AUTHOR	Schack, Ann; And Others
TITLE	Computer-Translation: Grade 2 Braille from Print. Final Report.
INSTITUTION	American Printing House for the Blind, Louisville, Ky.
SPONS AGENCY	Office of Education (DHEW), Washington, D.C. Bureau of Education for the Handicapped.
BUREAU NO	BR-6-1190
PUB DATE	Jun 69
GRANT	OEG-2-6-061190-1578
NOTE	98p.
EDRS PRICE	EDRS Price MF-$0.50 HC-$5.00
DESCRIPTORS	*Braille, Computer Programs, Computer Science, Educational Technology, Electronic Data Processing, English, Feasibility Studies, Input Output, Instructional Materials, *Machine Translation, *Material Development, *Mathematics, *Music, Programing

ABSTRACT

Two studies of computer production of mathematical texts and musical scores in Braille analyzed production practices and input preparation problems; the studies also reviewed the Nemeth Code of Braille Mathematics and Scientific Notation and the Revised International Manual of Braille Music. Both studies demonstrated automation to be feasible. Additional studies considered means of advancing computer translation of English Braille, including hyphenation, the 1968 program, proofreading, economic factors, and contraction contexts. A program abstract and a discussion of the potential of the IBM 360 series are included; also appended are a mathematical Braille translation system and a computer program to produce musical Braille. (JD)

ED036931

PA-40
BR-6-1190
OE/BEH

FINAL REPORT

Project Number 6-1190
Grant Number OEG 2-6-061190-1578

COMPUTER TRANSLATION: GRADE 2 BRAILLE FROM PRINT

Ann Schack, Joseph Schack,
Robert Haynes, and John Siems

The American Printing House for the Blind
1839 Frankfort Avenue
Louisville, Ky. 40206

June 1969

Department of Health, Education, and Welfare

U.S. Office of Education
Bureau of Education for the Handicapped

EC004963E

15

ED036931

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
OFFICE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY.

FINAL REPORT

Project No. 6-1190
Grant No. OEG 2-6-061190-1578

COMPUTER TRANSLATION: GRADE 2 BRAILLE FROM PRINT

Part I

Studies in the Automation of Braille Mathematics and Music

Ann Schack and Joseph Schack

Part II

Studies Toward Advancing Computer Translation of English Braille

Robert Haynes and John Siems

The American Printing House for the Blind

1839 Frankfort Avenue
Louisville, Kentucky 40206

June 1969

The research reported herein was performed pursuant to a grant with the Bureau of Education for the Handicapped, U.S. Office of Education, Department of Health, Education, and Welfare. Contractors undertaking such projects under government sponsorship are encouraged to express freely their professional judgment in the conduct of the project. Points of view or opinions stated do not, therefore, necessarily represent official position of the Bureau of Education for the Handicapped.

Department of Health, Education, and Welfare

U.S. Office of Education
Bureau of Education for the Handicapped

CONTENTS

Part I

STUDIES IN THE AUTOMATION OF BRAILLE MATHEMATICS AND MUSIC

I. The Scope of the Study 8
II. Background 9
III. Methodology 11
IV. Braille Mathematics 13
V. Braille Music 17
VI. Conclusions 21
References 22
Appendix
  A. Mathematical Braille Translation ... 23
  B. A Computer Program to Produce Musical Braille 69

Part II

STUDIES TOWARD ADVANCING COMPUTER TRANSLATION OF ENGLISH BRAILLE

Summary 81
Introduction 82
I. Space Requirements for Hyphenated versus Unhyphenated Braille 84
II. 1968 Braille Translation Program 87
III. Proofreading in Braille Production 91
IV. Economic Factors in Automated Braille Production 93
V. Abstract of Program for Braille Translation 95
VI. Potential of IBM 360 for Braille Translation 96
VII. Contexts Affecting Braille Contractions .. 98
Conclusions 100

PART I

STUDIES IN THE AUTOMATION OF BRAILLE MATHEMATICS AND MUSIC

Ann Schack and Joseph Schack

PREFACE

The authors acknowledge with gratitude the whole-hearted assistance of their colleagues at the American Printing House for the Blind: in particular, Mr. Robert Haynes and Mr. John Siems of the Data Processing Department, and Miss Marjorie Hooper, Mr. Ralph McCracken and Mrs. Nelle Edwards of the Editorial Department.

Thanks also are due Mr. James D. Keen, Senior Consultant, Bradford Computing and Systems, Inc., who assisted in the study of Braille Mathematics and Mr. Stefan Bauer-Mengelberg, President of the Mannes College of Music, who contributed his input language and his great knowledge of music to the study of Braille music.

Ann Schack
Joseph M. Schack
New York, N. Y.
1969

THE SCOPE OF THE STUDY

This report describes two studies designed to investigate the feasibility of using a computer to produce mathematical texts and musical scores in Braille. An IBM 709 computer has been used in the production of Braille literary material for several years. However, the rules which govern the transcription of mathematical and musical material are quite distinct from the literary rules and require completely new programs to apply them. Furthermore, the problems of preparing mathematical and musical material in a machine-readable form are very complex.

In the course of these investigations, the rules of THE NEMETH CODE OF BRAILLE MATHEMATICS AND SCIENTIFIC NOTATION 1965 (1) and the REVISED INTERNATIONAL MANUAL OF BRAILLE MUSIC 1956 (2) were studied in great detail. Consultations with personnel at the American Printing House for the Blind yielded information about current production practices. Studies of representative mathematical and musical material, both inkprint and Braille versions, highlighted the nature of the input preparation problems.

Both studies indicated the feasibility of automating the production of this material. The study of Braille mathematics progressed well into the development stage. A program plan was outlined and some sample subroutines detailed. Time did not permit the study of Braille music to progress to that extent, but a very general plan was developed and the discovery of a suitable input language solved the considerable problem of representing musical notation.

The authors conclude that computer programs can be written to produce Braille mathematics and music, and urge that the implementation of these programs begin as soon as possible. The shortage of skilled transcribers is acute. Automating the production of these specialized Braille codes will insure that the Braille reading population will continue to get the material they need and want for their educational and professional advancement.

BACKGROUND

Braille is a system of raised dot representation which enables the blind person to read using his sense of touch. The basic character is a six dot cell (⠿) and the 63 different characters are used, singly or in combination, to represent the total range of inkprint characters. This limited character set requires that each cell be multiply used. The significance of the characters and the rules governing their use vary, depending on the material represented. Literary material is transcribed according to the rules of ENGLISH BRAILLE AMERICAN EDITION (3) while music is written following the rules specified by the INTERNATIONAL MANUAL OF BRAILLE NOTATION. (2) The newest, most comprehensive system for mathematical material is the NEMETH CODE OF BRAILLE MATHEMATICS AND SCIENTIFIC NOTATION (1) currently undergoing revision prior to its final adoption.

Each of these codes defines the character set uniquely. For example, the Braille cell ⠦ (dots 2, 3, 6) may be used for the word HIS in a literary text. Properly used in a mathematical expression it is the number 8, and in music it is the symbol for the staccato, a dot above or below a note. Within each code, a given Braille sign may have multiple meanings. The rules specify how these signs must be used to avoid ambiguity. The literary code uses the same sign to represent an opening quotation mark, a question mark, and the contracted representation of the word HIS. For this reason, the word HIS must be spelled in full if it is in contact with a punctuation sign. The Nemeth Code, which uses the same punctuation signs, requires that a punctuation indicator be interposed between a number and a punctuation sign in order to distinguish the sequence 8? from 88. The rules of Braille music specify where the staccato symbol should be placed in relation to the note and other nuance symbols in order to insure clarity.

Each of these codes further details how material must be arranged, how information can be abbreviated or condensed, and how an expression or phrase can be divided if it is too long to be accommodated on one line. In the case of mathematical and musical material, it is obvious that format rules are extremely important, since the relative position of a number or note influences its significance. Such rules are also an important part of the literary code, especially in the representation of poetry, letters, lists, tables, and other special forms.

The great bulk of Braille published uses the literary code. In addition to fiction, poetry, non-fiction and textbooks, it is also used to transcribe the expository portions of mathematics and music texts, and lyrics for vocal music. It was therefore the first to be programmed for computer translation. A growing shortage of skilled Braillists, and the difficult and lengthy process of training replacements gave rise to the development of a computer program for translating literary material to Braille. That first program, written by IBM in cooperation with the American Printing House for the Blind, converts literary texts, in punched card form, into Grade 2 Braille, and produces an output deck of cards containing the Braille codes as well as format control codes. This deck operates the card-controlled stereograph equipment which produces the zinc master plates used to emboss Braille.

The original program, written for the IBM 704, (4) and the subsequent version for the IBM 709 (5) currently used in production have been extensively documented elsewhere and will not be detailed here. Briefly the basic unit of translation is a word, defined as a stream of characters between spaces. By reference to a highly specialized dictionary the program selects a tentative translation and then performs a series of tests to determine whether the Braille codes are correctly used. The program also arranges the Braille in the format required for the material being translated, supplies page numbers and titles, centers headings, etc.

Since 1964 when an IBM 709 computer was installed at the American Printing House for the Blind the original program has been considerably modified and expanded to handle a wide variety of material including textbooks which must be formatted according to a special set of rules. The computer has been used to produce more than 1000 Braille volumes.

The success of the literary translation program and the continuing difficulties in hiring and training transcribers led naturally to a consideration of the specialized codes. In 1966 the study of the Nemeth Code was undertaken to determine the feasibility of writing a computer program to apply those rules. A similar study of the Music Code was started in 1968. Although the methods used in these two studies are similar and the conclusions identical, the codes themselves and the problems encountered are distinct enough to warrant individual discussion. Therefore, following a general description of the methodology, the details of the mathematics and music studies are described separately.

METHODOLOGY

Determining whether any human activity can be performed by a computer requires a thorough analysis of the activity itself, as well as the nature of the data to be handled (input) and the form of the results (output). Although the study of these three aspects of the problem may be concurrent, the logical starting place is the processing which must be performed by the computer. The rules and practices which govern the manual activity must be examined to determine whether they can be codified for computer handling. Where there is a reliance on individual judgment for certain decisions, the attempt must be made to specify the basis on which such judgments are made.

Assuming that a preliminary investigation indicates that a computer can be programmed to make the necessary decisions, the problems of input and output must then be considered. If the input data does not already exist in a machine readable form it is necessary to develop a method for converting it. Similarly the computer produced output may have to be processed further to produce the desired results.

The input for any Braille translation program is printed material which, at present, must be copied by a keypunch operator. Future developments in optical scanning devices and the use of typesetter tapes may one day obviate this step but it is currently the most practical procedure. The problems of input preparation for Braille mathematics and music are similar to, but far more complex than those of literary Braille. The input language developed for the literary program specifies some special codes to represent inkprint characters not found on the keypunch keyboard and to indicate paragraph beginnings, chapter titles, and other characteristics of the printed material which can be represented in Braille.

Mathematical material, while utilizing the basic alphabetic and numeric character set, also includes numerous symbols for which there is no keypunch equivalent and the special formats are many and varied. One philosophy underlying the Nemeth Code is that the Braille representation be arranged as nearly like the inkprint as possible. It is therefore very important that the format of the printed material be completely and accurately indicated.

Musical material poses an even greater problem since almost none of the symbols are found on the keyboard, and the location of a given symbol, which is integral to its significance, may vary widely.

The essential requirements for an input language for mathematics or music, or any other special material, are that it be unambiguous and easy for an average keypunch operator to learn and use. The object in using a computer for Braille production is to utilize the readily achieved skill of keypunching instead of the very complex decision making required of a highly trained Braille transcriber. To require that the keypuncher be well versed in mathematics or music or Braille would negate the purpose of using the computer. An additional consideration in designing such a language is to minimize the number of keystrokes required to represent a given inkprint symbol and to select codes which, wherever possible, have some mnemonic significance.

The third aspect of the problem, that of output, is the one which required no additional development. Output from any specialized Braille program can be in the same binary card form currently used to control the stereograph equipment.

One further comment about methods is pertinent here. The depth of analysis and the developmental work, if any, will vary depending upon the nature of the problem, the amount of time allotted, and the kind of answer being sought. The two studies described here were designed to establish the feasibility of writing computer programs for these specialized Braille codes. Cursory study indicated that such programs are possible; the detail employed here is necessary to show that they are practicable.

The study of Braille mathematics occupied a major portion of the grant period, and progressed beyond a feasibility study into the design and partial implementation of an actual program. Further progress in that direction was interrupted, not only because the Nemeth Code was undergoing revision, but also to allow time to investigate the problems of Braille music. Although a relatively short time was spent in that study, two factors favorably influenced the amount of progress that was made. The formidable task of developing an input language for musical notation had already been accomplished as part of a project to automate music typesetting. The time spent studying that language to insure its suitability was but a small fraction of the time required to develop it. In addition, some of the problems of formatting music are analogous to those in mathematics and can be implemented using techniques similar to those developed during that study.

BRAILLE MATHEMATICS

Because mathematical texts contain both expository and mathematical material, the rules of the NEMETH CODE OF BRAILLE MATHEMATICS AND SCIENTIFIC NOTATION (1) were studied as a unit, and in relation to the rules of ENGLISH BRAILLE AMERICAN EDITION. (3) It was clear at the outset that any program for mathematical expressions would have to be coordinated with the existing literary program. It was also clear that the combined program should determine whether a given stream of input characters should be translated by the mathematical or literary portion of the program. Such a decision should not be required of the keypunch operator. These considerations were of prime importance in the early study of the problem.

A detailed review of the rules of Braille mathematics began first with many readings of the Nemeth Code and constant references to the rules of English Braille. The authors' familiarity with that latter code was quite valuable here. The Introduction to the Nemeth Code states that, in transcribing technical works, "the rules of English Braille and of the Nemeth Code will be rigidly followed within their respective domains." For the sighted transcriber, the boundaries of these domains are obvious from context. For computer interpretation, however, these must be defined in highly specific terms. Numeric information is, of course, easily identified. However, such alphabetic sequences as SIN x and COSH y are also mathematical expressions and must be transcribed according to the rules of the Nemeth Code, not English Braille. Identifying a mathematical expression is necessary to insure its correct transcription as well as the correct form of adjacent literary material. (e. g., Contractions for TO, INTO and BY must not be used before any mathematical expression.)

Deciding whether a computer could distinguish a mathematical expression was a necessary first step in the feasibility study. To this end, a representative selection of mathematical texts was reviewed to insure that an algorithm could be developed to handle the various forms of expressions. After considerable study, a general strategy was designed to satisfy the requirements discussed above: that the mathematical and literary programs be coordinated; and, that the computer be programmed to distinguish between these two domains.

The mathematics program was designed as a pre-processing routine which scans all input data before transmitting it to the existing Braille translation program. It performs no translation into Braille. Instead, it utilizes a small dictionary and a few analytic routines to distinguish mathematical expressions from expository text. The literary material is passed unchanged to the literary program to be translated according to the rules of English Braille. The mathematical material is modified by the addition of special codes which will cause the translation program to supply the appropriate Braille. The modification may be simply the insertion of special codes around the expression SIN x or the very complex adjustments necessary to format exercise arrays, computation examples, complex equations and the like. The existing program utilizes an expanded dictionary to perform all of the translation to Braille. This is especially desirable since the line arrangement and pagination functions are embedded in the present program. To separate the translation of the two types of material would involve unneccessary duplication.

In this kind of study, the analysis of the problem and the development of solutions are concurrent and interactive processes. While detailed solutions were not considered until later in the study period, the over-all design and some specific techniques were being developed as each new aspect of the Nemeth Code was studied.

Once the basic structure of the pre-processor was outlined, attention was focussed on the not inconsiderable problem of format. Dr. Nemeth states that "the Code is intended to convey as accurate an impression as is possible to the Braille reader of the corresponding printed text." (6) It is therefore necessary to supply as input to the computer an accurate description of the content and the arrangement of the printed page. The description should be in terms which require no specialized mathematical training and require as few keystrokes as possible.

Developing this input language required studying, not only the rules of the Nemeth Code, but also the practices currently used to transcribe mathematical texts. Because elementary school texts are an important segment of the mathematical Braille produced at the Printing House, attention was directed to the format problems found there. It is interesting to note that, because of the introductory nature of these books, there are more unusual spatial arrangements than would be encountered in higher level texts. With the help of the Mathematics Editor, Mr. Ralph McCracken, a representative group of texts was selected for study. These were books which included examples of 'conventional' as well as 'new math' presentations. Because they had already been transcribed manually, the inkprint copy which had been annotated by the editor and the Braille edition were both available.

In the course of studying this material, a general plan for an input language was developed and specific codes and statements were designed. The assignment of special codes to represent characters not on the keyboard was based primarily on considerations such as mnemonic value and keying ease. The programming required to translate these is trivial. However, the choice of format statements was influenced by programming considerations as well as ease of use.

In general, the more information supplied by the keypunch operator, the less work required of the computer. Consider, for example, a set of addition exercises shown in print as follows:

	a	b	c
1.	15 3	10 12	6 2
2.	22 11	5 9	17 21

In Braille, the arrangement of the exercises should be the same if space permits. However, the column designation (a, b, c) is shown next to each exercise, followed by a period, the separation lines are extended, the problems must be separated by two spaces, the column and row designations by two spaces, etc. The keypunch operator could be asked to key the column headings appropriately, and to supply the spaces as required by the Braille rules. The computer would select the Braille codes and supply numeric and punctuation indicators as required. However, such an approach requires of the keypunch operator almost as much work as is performed by the Braillist and would make the use of the computer impracticable.

In order to minimize the work required of the keypunch operator, a simple descriptive statement was designed. The work of rearranging, inserting periods, counting spaces, aligning exercises is performed by the computer. The exercise array shown above is described as follows:

XFORM 2 rows (1.-2.) 3 cols (a-c)

Each example is then punched as a unit including the separation line. (e. g., 15,3// 10,12// 6,2// etc.) The XFORM statement provides the information needed for the computer to arrange the exercises properly.

Each statement was designed to meet two requirements; to provide all the necessary information to the computer; and, to minimize the work and special knowledge required of the keypunch operator. It was then tested by several operators at the Printing House who keypunched specified portions of a text. The results of these tests and the operators' comments provided a basis for modifying the statement where necessary, or for expanding the instructions about its use. Some examples of these statements and instructions are found in Appendix A, as well as a flow chart and detailed description of a routine to implement the XFORM statement.

The set of statements which were designed and tested during this study is by no means a complete input language for all mathematical material. It is probably sufficient for most elementary school texts. However, the program has been designed to facilitate the addition of subroutines to handle new format statements as they become necessary.

This section describes the major problems which had to be solved to demonstrate the feasibility of writing a program to produce Braille mathematics. The methods of studying these problems and some proposed solutions have also been described in general terms. A detailed discussion of these and many other aspects of the Nemeth Code and the proposed program is found in Appendix A.

Because of the depth of this study, the findings can be stated very positively. A program has been designed which demonstrates that a pre-processor can be linked with the existing Braille program in use at the American Printing House. It can distinguish mathematical expressions, supply the many new indicators required by the Nemeth Code, insure the appropriate use of contraction codes - within or in contact with mathematical expressions, and can make use of a minimum amount of information about format to produce properly arranged Braille. Although the present design was intended to work with the existing 709 program, the validity of the feasibility statement is not dependent upon either the computer or the program. It has been demonstrated that the rules of the Nemeth Code are 'computable' and that complex spatial arrangements can be described using a set of fairly simple statements.

BRAILLE MUSIC

As was true for the study of Braille mathematics, the two major problem areas considered during this study of Braille music were the preparation of input, and the processing required of the computer. However, the nature of the problems vary in degree and kind. The difference in the input preparation problems is obvious. Describing musical notation using a keypunch requires an extensive input language much larger than the set of input statements and codes required for mathematical material. There is very little coincidence between inkprint music symbols and the character set of the keypunch. Only lyrics, time signatures and other numbers can be transcribed directly.

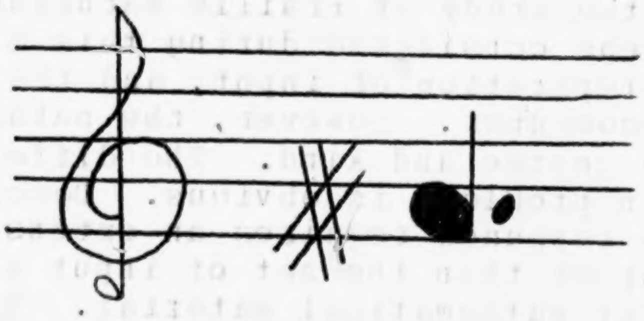
The second major difference is exemplified by the following quotes:

> "The rules of the code are intended to require a minimum of decision making and interpretation on the part of the transcriber... the transcriber is required to represent the sign and to be unconcerned about the meaning of that sign."
>
> NEMETH CODE OF BRAILLE MATHEMATICS (6)

> "This book has been written for the use of seeing musicians who wish to learn the system of Braille music notation in order to transcribe inkprint music into Braille.... The student should have a complete knowledge of the rudiments of inkprint staff notation, together with special facility in whatever type of music interests him as a transcriber."
>
> LESSONS IN BRAILLE MUSIC (7)

A further aspect of the difference between the two codes is the degree of choice permitted the transcriber. Dr. Nemeth specifically directs that the rules be followed literally and enjoins the transcriber against making any modifications. In contrast, the INTERNATIONAL MANUAL OF BRAILLE MUSIC, a compilation of decisions reached at the 1954 Paris Conference, strives to strike a balance between "rigid regimentation of rules and an easy acceptance of alternatives." (8) The latitude permitted by the Code, and the musical knowledge required of the transcriber appear, at first, to be serious obstacles to the use of a computer to produce Braille music. However, closer examination revealed that a computer can be programmed to handle much of the decision making currently required of the transcriber.

The musician-transcriber, familiar with inkprint staff notation, recognizes that a note represented in inkprint as follows:



is an F sharp, dotted quarter note. For purposes of Braille transcription it must also be noted that the note is in the fourth octave. This is an objective decision based on the color and shape of the note, its location on a staff, and the definition of the staff as indicated by the clef sign. Given these same data a computer can be programmed to identify the note.

The problem is to develop a means of describing all these objective characteristics of a given musical symbol. The language should be easy to learn and apply, and should require no special musical training. The number and variety of musical symbols is such that developing such a language is a major task. During the period of this study the authors could have made only a small contribution. Fortunately, such a language had already been developed as part of a project to automate musical typesetting. Mr. Stefan Bauer-Mengelberg and his colleagues, working under a Ford Foundation grant, spent nearly three years designing and testing a method of keypunching musical notation.

Mr. Bauer-Mengelberg not only made this language available, but served as consultant to the study. The language uses symbols, wherever possible, which have mnemonic significance. The duration of a note, indicated by its shape and color, is indicated by a single alphabetic character. Its location is described by a one- or two-digit number which refers to the line or space on which it appears. The lines and spaces of the G clef staff are numbered 20 - 29. Because these notes are so frequently used, the language permits the abbreviation of these codes by eliminating the 2. Each note, then, is described in terms of its location and shape. Other symbols which influence its musical significance, accidentals, articulation symbols, etc., are punched in prescribed sequence adjacent to the basic description. The note shown in the example would be punched:

2#Q.

The motivation for automating musical typesetting and musical Braille are remarkably coincident. Both projects were undertaken because of the growing shortage of highly skilled technicians - typesetters and Braillists. Because this language was designed to represent the printed page with great accuracy, it can serve perfectly as input to a Braille translation program, thus eliminating the need for the knowledge of musical notation now required.

However, the other major problem is the freedom accorded the transcriber to choose among alternate methods of representing a particular passage. The most obvious example is the choice of three methods of presenting keyboard music - bar over bar, line over line, and section by section. According to LESSONS IN BRAILLE MUSIC, "the layout is really very much the same in all three." However, this manual discusses "the rival claims of each of these systems ... in order to help the student in making a suitable choice of disposition to suit different types of music." (9) Here the transcriber is basing a decision on musical judgment and personal opinion about what will simplify the task of the Braille reader. Such judgments, not based on objective criteria, are often made differently by two transcribers.

No computer can be programmed to make such judgments. However, two solutions are possible. Whichever presentation is chosen, the resultant Braille music is correct and complete. The choice exists because an agreement on one of these methods could not be reached at the 1954 International Conference on Braille Music. One solution is to adopt one of these methods as the standard to be used for all computer produced Braille music. The program would always present keyboard music in the same format. An alternate plan would require an editorial decision about each piece of keyboard music. The computer would be programmed to apply the method specified. Even though this requires some manual intervention, the monumental task of transcribing and arranging the Braille is assigned to the computer.

Some other choice points faced by the transcriber have to do with adding to or subtracting from the inkprint representation. Such modifications are made to clarify the music for the Braille reader, to simplify the task of memorization and to cut down on the bulk of the finished work. For example, the transcriber may on occasion add rests, accidentals, or other symbols whose meaning is inferred by the sighted musician because his eye can take in several staves at once. For the Braille reader, concentrating on one staff at a time, adding these symbols is important.

The transcriber can also eliminate certain note sequences by the judicious use of repeat signs. There are many rules and restrictions concerning the use of these repeat signs and most of them can be stated in a form suitable for computer application. In some cases, there is a choice among several methods of presentation which is left to the individual transcriber's judgment. Especially with regard to the use of the part-measure repeat, its use is a "never-ending study, and even among experts opinions vary very much in detail." (10) This problem is analogous to the choice of score disposition discussed earlier and the same set of solutions are clearly applicable.

Although time did not permit the program design and development which was part of the study of Braille mathematics, the findings of this study of Braille music can be stated just as positively. The task of developing an input language is a major one - and has already been solved. Despite the leeway currently permitted the transcriber, it is clearly possible, and perhaps desirable, to make certain arbitrary decisions which can be programmed. Complex music will undoubtedly require editorial review, but the annotation required will be far less than that of the sometimes extensive rewriting required for mathematical texts.

Finally, the great variety of inkprint music, while it requires extensive programming, can be segmented in a way to facilitate the building of a complete program. The program can be designed in such a way that sub-sections to handle different musical forms can be added gradually. Monophonic music can be implemented first, then simple polyphonic forms, and the program can be progressively elaborated to handle more complex music and such special forms as music for plucked instruments, accordion, etc. This would be analogous to building the shell of a house, and then completing one room at a time. Such a design insures some utility at the earliest possible date, and provides a means for expanding the system without disturbing the already completed parts.

CONCLUSIONS

The studies of Braille mathematics and music have shown that the rules of each code can be applied by a computer program with a minimum of editorial intervention. The problems of describing the content and arrangement of the printed page have been investigated and solutions have been developed. It is clear from these studies that computer programs can be written to produce both Braille music and mathematics which will conform to the quality standards of the American Printing House for the Blind.

In light of the growing shortage of skilled transcribers, and the continuing and expanding needs of the Braille reading population, the conclusions drawn here should not be treated academically. Implementing such programs is a lengthy procedure and the completed programs must be tested on a wide variety of material. Experience with the literary program has demonstrated this, and has also demonstrated the value of using a computer for Braille production. It is the authors' observation that the skill required for transcribing music is more complex and in shorter supply than that required for producing elementary mathematics texts. If this is so, the music program should be undertaken first.

Which program should be implemented first, or whether they should be undertaken simultaneously is, of course, an administrative decision. Because of the time and skill required to produce a working program, it is the authors' recommendation that a decision be made at the earliest opportunity, and that implementation be started before the shortage of transcribers becomes a lack of transcribers.

REFERENCES

1. THE NEMETH CODE OF BRAILLE MATHEMATICS AND SCIENTIFIC NOTATION, 1965, (Louisville: American Printing House for the Blind, 1966)

2. Spanner, H.V. (compiler), REVISED INTERNATIONAL MANUAL OF BRAILLE MUSIC NOTATION, 1956, (Louisville: American Printing House for the Blind, 1961)

3. ENGLISH BRAILLE AMERICAN EDITION, 1959 (Louisville: American Printing House for the Blind, 1962)

4. Schack, Ann, and Mertz, R.T., BRAILLE TRANSLATION SYSTEM FOR THE IBM 704, (New York: IBM Corp., Mathematics and Applications Department, 1961)

5. Part II, p. 87, of this report

6. THE NEMETH CODE, p. x

7. Spanner, H.V., LESSONS IN BRAILLE MUSIC, (Louisville: American Printing House for the Blind, 1961)

8. MANUAL OF BRAILLE MUSIC, p. viii

9. LESSONS IN BRAILLE MUSIC, p. 51

10. ibid., p. 40

Appendix A

MATHEMATICAL BRAILLE TRANSLATION SYSTEM

TABLE OF CONTENTS

A. INTRODUCTION

A brief review of the rules of Grade II English Braille and a discussion of the rules of the Nemeth Code.

B. THE ROLE OF A COMPUTER

A general discussion of the computer techniques used to apply both sets of Braille rules.

C. THE MATH TRANSLATION PROGRAM

A flow chart and description of the program.

D. TABLES

A description of three main tables, their functions and the search techniques used.

E. TYPICAL SUBROUTINES

Detailed descriptions of several types of subroutines, including input handling, formatting, and indicator insertion.

F. KEYPUNCH INSTRUCTIONS

G. SUMMARY

A. INTRODUCTION

Braille is a system of embossed representation of printed material based on a six dot character, or cell. The significance of these 63 cells, and the rules governing their use are defined by several different Codes depending on the nature of the material being transcribed. The computer program described in this report is designed to translate elementary school mathematical texts into Braille, according to the rules of English Braille, Grade II and the Nemeth Code of Braille Mathematics. A brief review of these two Codes is included here as background for the discussion of the program which follows. (For more detailed information, see ENGLISH BRAILLE AMERICAN EDITION, 1966, and THE NEMETH CODE OF BRAILLE MATHEMATICS AND SCIENTIFIC NOTATION, 1965.)

The rules of English Braille assign a cell or pair of cells to represent the letters of the alphabet, the ten digits, punctuation signs, composition signs, Braille indicators, and certain letter combinations. Some cells have multiple significance and the Code includes rules governing their use. For example, the ten digits are represented by the Braille codes A through J. To distinguish the numeral from the letter, the numeric indicator must precede a series of numbers. Similarly, in a mixed stream of alphabetic and numeric material, the alphabetic indicator must precede the letters A through J to avoid confusion.

Some cells stand for one or more letter combinations as well as a punctuation sign. For example, the same cell is used to represent BE, BB, and the semi-colon (;). Here, confusion is avoided, not by the use of indicators, but by a set of rules which specify the conditions under which the contraction code may be used. In this case, the letters BE may be contracted only for the whole word or the first syllable of a word. The BB combination may be contracted only in the middle of a word. Thus, the BE contraction may be used in BESIDE, but not in ADOBE. Similarly, the BB contraction may be used in RUBBER, but not in EBB. Violation of these restrictions would cause ADOBE to be misread as ADO; and EBB as E;.

The use of all contractions, including those with no multiple significance, is specified by rules designed to facilitate the reader's recognition and pronunciation of a word. Thus, specifically, a contraction may not be used "where it would violate the primary division between a prefix and the base word", and generally, contractions may not be used "where they would obscure the recognition or pronunciation of a word." The use of short-form words, or abbreviations is similarly restricted. The abbreviation for

BLIND (BL) may not be used in the word BLINDED, since it would be misread as BLED.

In addition to rules about how words should be transcribed, there are carefully specified format conventions. They govern the size of pages, page numbering, paragraphing, the arrangement of chapter headings, the spacing of poetry, etc.

The rules of English Braille are used for transcribing literary, non-technical material - whether it is accomplished manually, or by the 709 computer program. Some of the techniques used by that program to apply these rules are described in the succeeding section.

The Nemeth Code of Braille Mathematics specifies how mathematical and scientific notation is to be represented. Since all textbooks contain some expository sections, the Nemeth Code is used in conjunction with the rules of English Braille. In general, these rules must be used in transcribing the literary portions of the text. The exceptions are specified by the Nemeth Code.

While it uses the English Braille code assignments for letters and punctuation, the Nemeth Code introduces a new set of codes for numerals, additional indicator codes, and of course, new code combinations to represent mathematical symbols not found in literary texts. Numerals are represented by codes A through J in the lower part of the cell. Thus the letter B is still dots 1 and 2 but the number 2 is dots 2 and 3. The rules for the use of the alphabetic indicator are changed because there is no longer any confusion between the letters and the numerals. However the lower cell numerical codes can be confused with some of the punctuation symbols. For example, the lower cell B (number 2) is the semi-colon. For this reason a punctuation indicator is specified by the Nemeth Code, to be used when certain punctuation signs are associated with mathematical expressions. The Nemeth 4, for instance, is the same code as the period. Therefore the sequence 123. requires the punctuation indicator preceeding the period to distinguish it from the sequence 1234. The numeric indicator is still necessary because the lower cell Nemeth numerals can be confused with certain whole word contractions specified by English Braille. Consider the sentence 'There were 7 balls'. The Braille code ⠶ stands for the whole word contraction for 'were' as well as the Nemeth 7.

The Nemeth Code restrictions on the use of contractions are also based on the multiple meanings of the lower cell symbols. The Nemeth rules state that the contractions for 'to, into and by' may not be used before a mathematical expression. The reason for this becomes obvious when one realizes that the contraction for 'to' is the same as the Nemeth 6. Permitting the use of the contraction in such cases would lead to confusion. Similarly 'into' might be read 96 - 'by' as 0.

The introduction of new Braille symbols for certain punctuation signs is again based on the possibility of confusion with the numerals. The literary Braille symbol for the parenthesis is the Nemeth numeral 7 and the literary comma is the Nemeth 1. To avoid such reading problems the Nemeth Code includes new symbols for the mathematical comma, mathematical parentheses, brackets, etc.

Other aspects of the Nemeth Code are independent of the literary code and are based solely on the nature of the mathematical material as presented in inkprint. The Code specifies new symbols and combinations to represent such signs as arrows, comparison signs, logical symbols, and so on, which do not occur in literary texts. There are also new indicators introduced to facilitate reading comprehension. The fraction indicator provides a signal to the reader that the succeeding material is arranged above and below a fraction line and influences his method of scanning the following codes.

An important philosophy underlies the large segment of the Nemeth Code dealing with spatially arranged material. The attempt is to make the Braille presentation as equivalent to the inkprint as possible. The number and variety of the rules are quite extensive. Some of these are discussed in the following sections.

B. THE ROLE OF THE COMPUTER

The several categories of English Braille rules are implemented by the computer program using two primary techniques: a detailed scan of the input word and comparison of adjacent characters or bites; and, reference to a stored dictionary to locate whole words or parts of words. There are two types of entries in the dictionary. The first of these governs the use of contractions and supplies the appropriate Braille codes for output. A smaller group of entries is used to arrange the Braille codes in the desired format. These entries contain 'pseudo-codes' which are analysed by the program and converted to spaces, line endings, page endings, etc. where necessary.

Since textbooks contain both literary and mathematical material, they must be converted to Braille according to the rules of English Braille as well as the rules of the Nemeth Code. Where there are discrepancies, it must be determined which set of rules should be followed. Just as a skilled Braillist must 'know' both sets of rules, so must the computer program be able to apply both properly.

In order to achieve this, the existing Literary Braille program, with an expanded table, will be utilized as the Translator. Input to the Translator will be provided by the Math Pre-processor which will scan keypunched input, determine which set of rules should be followed, and will provide appropriate signals to the Translator.

The expanded table in the Translator program will accommodate the enlarged code set specified by the Nemeth Code and will include entries which govern the translation of pseudo-codes supplied by the Pre-processor. This table, and the code set, is described in detail in Section D. With that exception, this document describes the structure and function of the Math Pre-processor.

Keypunched input to the Pre-processor is analysed by the program to determine whether it is literary or mathematical. Depending on the nature of the material, the Pre-processor will function in one of three different ways:

1. If the word is literary, the input codes will be moved to the input list used by the Translator. The program has been designed to make this identification as quickly as possible in order to speed translation.

2. Mathematical material will be analysed to determine whether additional codes are required and whether the input codes must be converted. This analysis is performed by reference to a table of significant mathematical words, or by a detailed scan of a sequence of codes which includes some mathematical symbol. The stream of characters which is then passed to the Translator is an expanded and converted version of the original input.

3. When a Format code is encountered, the usual word-for-word transmission from the Pre-processor to the Translator is interrupted. The Pre-processor will continue to collect input words until the appropriate terminating code is read. When the intervening material has been arranged, by inserting line-ending codes and spaces, the Pre-processor will pass it, one word at a time, to the Translator.

The techniques used by the Pre-processor to apply the rules of the Nemeth Code are similar to those used by the Translator. There is a difference in emphasis, of course, determined by the differences between the Literary and the Nemeth Codes. For example, the Translator performs a bite-by-bite scan to determine whether a given contraction may be used, or whether one of two indicators (numeric, alphabetic) should be inserted. The scan performed by the Pre-processor also determines whether an indicator is required, but the larger set of indicators specified by the Nemeth Code means that there are more insertion routines and they will be used more frequently. The second function of the scan is to chose between the mathematical or literary representation of certain ambiguous inkprint signs. In certain of these cases, the comparison of adjacent characters may not provide definitive information and the Pre-processor will extend the scan to succeeding words.

Some of the Nemeth Code rules governing the non-use of contractions are implemented by using a table. Unlike the Translator's Grade II Table, which is contraction oriented, the Pre-processor's Word Table defines a small group of words which cannot be contracted. Both tables include special format codes supplied by the keypunch operators. The difference in the method of processing these codes is determined by the wide variety of spatial arrangements used for mathematical material in contrast to the more limited arrangements permitted for literary material. Therefore, associated with each format input code is a subroutine which will accomplish the specified arrangement.

The Math Braille Translation Program is a single program, although the Pre-processor and the Translator are described separately in this document. The succeeding Sections describe in further detail the tables and routines used by the Pre-processor to supply the Translator with information necessary to produce a properly coded mathematical textbook.

C. THE MATH PRE-PROCESSOR

The attached flow charts illustrate the general plan of the Pre-processor program. Output from this program becomes input to the Translator, which performs the conversion to Braille. When the input has been identified as literary, the Pre-processor passes the original input codes to the Translator.

In handling mathematical material, the Pre-processor may either insert indicator codes, and/or, convert the input codes to a pseudo-code form which will influence the Braille translation. For example, the input code 22_8 represents the letter 'B', which is converted to Braille dots 1-2 by the Translator. According to the rules of English Braille, this letter, standing alone, must be preceded by the alphabetic indicator to distinguish it from the word 'BUT'. If the letter is in contact with a numeral, it must be preceded by the indicator to distinguish it from the numeral '2'.

Since the Nemeth Code numerals are represented by lower cell codes, there can be no confusion between '2' and 'B', and the indicator is not required. Similarly, the indicator is not required in the statement 'a b+c'. Therefore, when a single letter occurs in a mathematical statement, it is converted to a pseudo-code (B=122_8) which produces the correct Braille code and prevents the insertion of the alphabetic indicator.

The following examples will serve to amplify the flow charts. They illustrate how the program handles alphabetic material which is literary, alphabetical material which is mathematical, punctuation and numerical input.

Example 1:

Divide 400 by 2.

The input codes for the first word in this sentence are:

=divide

(The = is the symbol for the capitalization indicator supplied by the keypunch operator.) The analytic scan performed by the Pre-processor finds that there is no mathematical symbol here, but that there is at least one non-alphabetic character. The program then follows the steps outlined under ALPHA AND PUNCT. P-SCAN strips off indicators and punctuation so that the alphabetic characters alone may be examined. Since this word is not found in the Table, it is passed directly to the Translator.

The next character stream,

400

follows the SOME MATH branch of the program. The function of M-SCAN is to determine the need for insertion of indicators or the selection of the mathematical symbol for certain ambiguous punctuation signs. In this case, M-SCAN supplies the pseudo-codes which produce the Nemeth Braille numerals, and which will cause the Translator to insert the numeric indicator at the beginning of the sequence.

The word,

by

is located in the Special Word Table, and is placed in the HOLD stack, pending the analysis of succeeding input.

The final sequence,

2.

is processed by M-SCAN, which supplies to pseudo-code for '2' and inserts the punctuation indicator in front of the period.

The next step, setting the Grade I switch, is accomplished by passing the codes,

$IG1

to the Translator. Until its effect is terminated by the codes $TG1, this causes the Translator to use only the Grade I Table, thus avoiding the use of contractions.

Since the HOLD stack is not empty, the OUTPUT routine puts out:

by 102|

(102 Nemeth 2; |=punctuation indicator)

Because the Grade I switch is on, the lower cell contraction for 'by' will not be used. Had the sequence been 'by land', the Grade I switch would not be turned on, and the contraction would be used.

Example 2:

(sin x)

The first set of codes,

(sin

turns on the P-Switch and, after P-SCAN, the unpunctuated word is located in the Special Word Table. Since an abbreviated function name is a mathematical expression, and must be punctuated properly, the entire sequence is analysed by the M-SCAN. The Math-Switch setting causes M-SCAN to supply the pseudo-code for the mathematical parenthesis. Not detailed in the flow chart is that fact that, when an open grouping symbol is read, M-SCAN will place words in the HOLD stack and continue to fetch input until the matching closing symbol is read.

Therefore, M-SCAN reads the next sequence,

x)

and selects the appropriate mathematical code for the parenthesis. In addition, it converts the 'x' code to the form which will prevent the insertion of the alphabetic indicator by the Translator. The Grade I code which precedes the entire sequence prevents the use of the 'in' contraction in the word 'sin'.

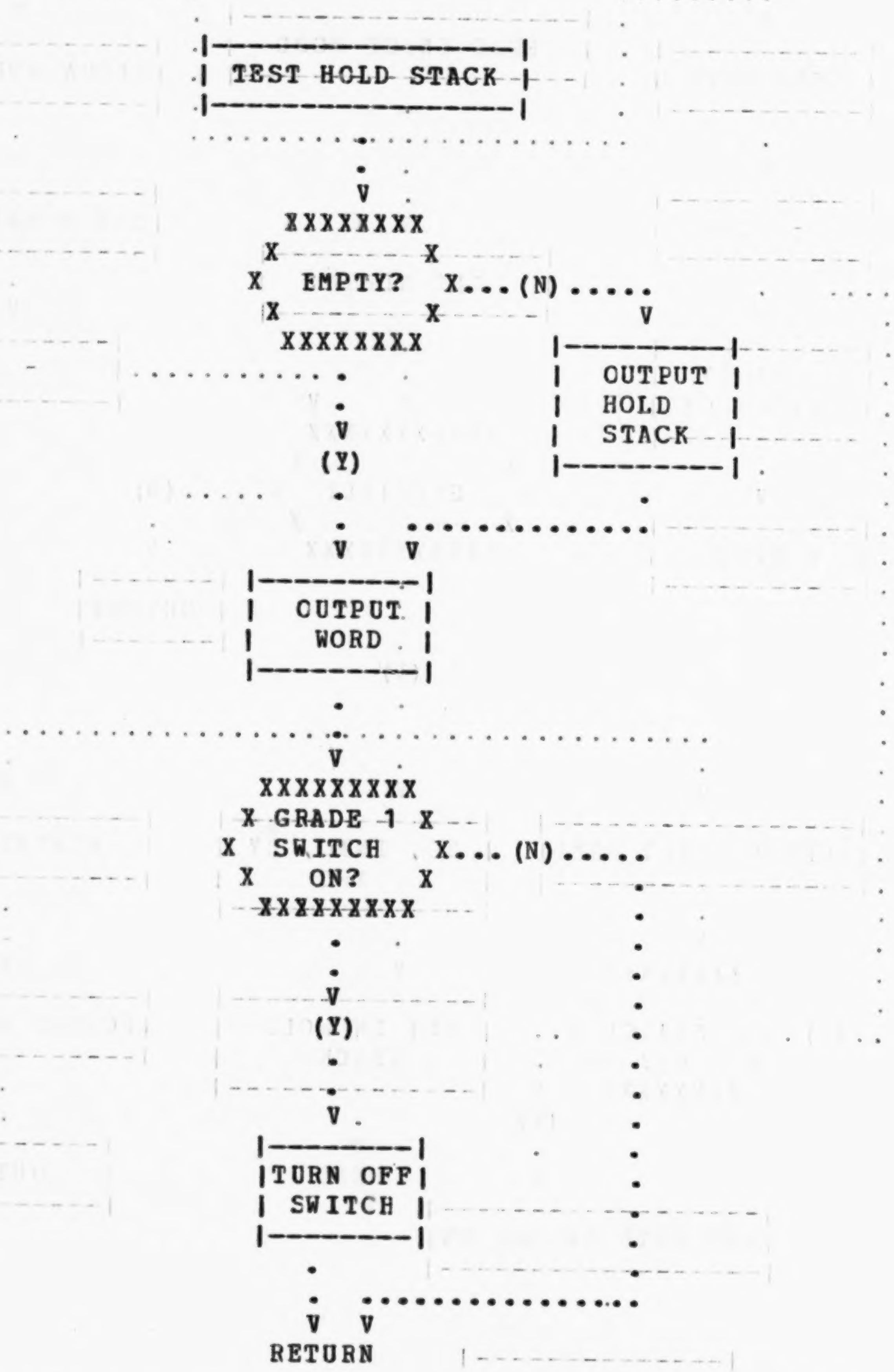
Since much of the material in mathematical textbooks is literary, the program has been designed to identify non-special alphabetic input quickly, in order to speed the translation process. The pre-processing required for mathematical sequences is offset by the use of the Grade I Table to translate these - a relatively fast procedure.

The tables and routines named here are described in further detail in the succeeding sections.

GENERAL FLCW CHART - MATH PRE-PROCESSOR

```
       .........                       ...........
       V       . |-------------------| .         V
 |-----------| . |  READ INPUT WORD  | . |---------------|
 | SOME MATH | . |-------------------| . |ALPHA AND PUNCT|
 |-----------| .           .           . |---------------|
       .       .........................         .
       V                   .                     V
 |-----------|             .             |---------------|
 | M - SCAN  |             V             |SET P SWITCH ON|
 |-----------|       |---------------|   |---------------|
....   .             |   ALL ALPHA   |           .
.  V   V             |---------------|           V
.|-----------|             .               |-----------|
.|  SET G1   |             .    ...........|  P - SCAN |
.| SWITCH CN |             V   .V          |-----------|
.|-----------|        XXXXXXXXXXX
.      .             X           X
.      V            X   SPECIAL?  X.....(N)
.|-----------|       X           X       .
.|  CUTPUT   |        XXXXXXXXXXX        V
.|-----------|             .          |-------|
.                          .          | OUTPUT|
.                          .          |-------|
.                         (Y)
.                          .
.         ........................................
.         .                .                      .
.         V                V                      V
.|------------------|  |---------------|   |---------------|
.|ABBREV FUNCT NAME |  | TO, INTO, BY  |   |  FORMAT CODE  |
.|------------------|  |     ARC       |   |---------------|
.         .            |---------------|          .
.         V                .                      .
.      XXXXXXXX            V                      V
.     X        X       |---------------|   |---------------|
..(N).X P SWITCH X..   | PUT IN HOLD   |   |FORMAT ROUTINE |
      X   CN?  X  .    |    STACK      |   |---------------|
       XXXXXXXX   V    |---------------|          .
                 (Y)        .                     .
                  .         V                |-----------|
                  V        READ              |  OUTPUT   |
      |------------------|                   |-----------|
      |SET MATH SWITCH ON|
      |------------------|
               .
               V
        |---------------|
        |   M - SCAN    |
        |---------------|
```

OUTPUT SUBROUTINE


TEST HOLD STACK
EMPTY?
(N)
OUTPUT HOLD STACK
(Y)
OUTPUT WORD
GRADE 1 SWITCH ON?
(N)
(Y)
TURN OFF SWITCH
RETURN

NOTE: WHEN CALLED BY FORMAT ROUTINES, RETURN IS TO THE FORMAT ROUTINE UNTIL LAST WORD IS OUTPUT; THEN, TO READ

D. TABLES

The Math Translation System makes use of three tables to apply the rules of the Nemeth Code. Two of these are used directly by the Pre-processor and will be referred to as the Symbol Table and the Word Table. The third is an expanded Grade I Table which is part of the Translator program.

Expanded Grade I Table

The original Grade I Table was designed to translate all the valid input characters to the appropriate Braille cell. The two entry words for each character contain the Braille code and the rules governing the translation. In order to accommodate the additional symbols specified by the Nemeth Code, this table is expanded to include codes for which there is no input character (Braille indicators) and codes for which there are two different representations depending on context (numerals, certain punctuation marks, etc.). Since input to the Translator is provided by the Pre-processor, the codes need not be limited to six bits but will use a seventh and eighth bit to provide the additional information.

This expanded code set not only provides additional symbols, but also is used to avoid redundancy between the functions of the Pre-processor and the Translator. For example, the Translator program inserts two indicators (numeric and alphabetic) according to the rules of English Braille. By providing the appropriate eight-bit pseudo code, the Pre-processor will cause the Translator to insert these indicators according to the rules of Nemeth Code as well.

For example, a sequence of numerals may be represented in three different forms in a mathematical text:

1. Numerals which represent page numbers are preceded by the numeric indicator and the Braille codes A-J stand for the numerals 1-0.

2. Numerals which are mathematical statements within the text are preceded by the numeric indicator and are represented in Braille by the LOWER CELL codes for A-J.

3. Numerals arranged for computation are not preceded by the numeric indicator and are represented by the LOWER CELL A-J codes.

The Pre-processor will perform the analysis necessary to determine which of these representations is correct and will supply the proper one of three possible numeric input codes.

In the first case, a reference to 'page 13' in the text would be interpreted by the Pre-processor which would supply to the Translator the codes 001, 003. The table entries for these codes contain the Braille codes for A and C, and the rules entries will cause the insertion of the numeric indicator in front of the 1.

The same numbers in the sentence '13 is a prime number' would be converted to the pseudo codes 101 and 103. The table entries for these codes contain the lower cell Braille codes, but the rules would still cause the insertion of the numeric indicator.

Should the same number appear as follows:

```
 13
 x2
 26
```

the Pre-processor would supply the pseudo codes 201 and 203. The Braille codes in these entries would again be the lower cell codes, but the rules associated with these entries would not cause the Translator to insert the numeric indicator.

The insertion of the alphabetic indicator is handled analogously. A reference to 'page 2A' requires the insertion of the alphabetic indicator, but the indicator is not required for the statement '2A + 3B = 13'. The proper conversion is accomplished by the use of pseudo codes for the letters A-J when they are in contact with lower cell numerals, or when other aspects of the context rule out the use of the indicator.

Symbol Table

This table contains an entry for each input code produced by the keypunch and each entry word is divided into three segments, as illustrated below:

```
AM   Description              Subroutine
/--/-/---------------/---/---------------/
S1   3               17   21              35
```

Bits S and 1 describe whether the character is alphabetic or mathematical. The Sign bit is 0 for alphabetic characters, 1 for all others. Bit 1 is 1 for all mathematical characters (numerals, operators, etc.) and 0 for all others. Thus, the letter A is coded 00, the number 3, 11 and the comma, 10. (Note: As an input code, the comma is neutral; the Pre-processor determines whether it is mathematical or literary.) These bits are used to set a switch which describes the nature of a word or character stream. The bits for each character are combined (using the OR instruction) to produce one of three different settings:

CAT	C (00) * A (00) * T (00) = 00	All alphabetic
2A	2 (11) * A (00) = 11	Some math symbol
AT,	A (00) * T (00) * , (10) = 10	Alphabetic and punctuation

As these bits are defined, no character can be coded 01. The final code, or switch setting, determines how that word is handled by the Pre-processor. Words which are all alphabetic (00) are looked up in the Word Table. A sequence of characters which includes at least one mathematical symbol is scanned to determine whether indicators should be inserted or whether input codes should be converted. In the case of punctuated alphabetic material, the alphabetic portion must be isolated before reference to the Word Table.

The Decrement, bits 3-17, are used to describe certain characteristics of the symbol. These categories are not exclusive, and several bits may be used to describe one character. For example, a left parenthesis would be described as:

> Open grouping symbol, ambiguous punctuation, initial punctuation.

The designation, Initial Punctuation, is used to facilitate the separation of alphabetic material from associated punctuation before referring to the Word Table. The Ambiguous Punctuation bit means that the subroutine must select one of two codes depending on context. The Open Grouping Symbol bit would be used if the mathematical (or literary) nature of the enclosed material is not immediately obvious. Under certain conditions the program will scan for the Final Grouping Symbol in order to select the proper code.

The Address portion of the word, bits 21-35 contain the address of the subroutine which will convert the input character when necessary, insert indicators or spaces, or pass it, unchanged, to the Translator.

Information in this table is arranged so that it can be retrieved quickly and simply. The descriptive bits for a given character are stored in a location whose address is computed by subtracting the input character bits from the Symbol Table Origin. Thus, the indicator bits and subroutine address for the number 5 are found in location (Table Origin - 5). The character itself is not stored in the table. The code to be output is either retrieved from the input stack or supplied by the subroutine. When necessary, the input code may be modified before it is passed to the Translator.

Word Table

This table contains the alphabetic codes and associated indicator bits for three types of words:

1. Format codes which identify and describe information which is to be arranged spatially. The subroutines which handle these codes will collect input until the appropriate terminating code is read, will insert spaces and line ending codes where necessary, and then pass this material to the Translator.
2. Abbreviated function names (SIN, COS, etc.). These are included in the table so they can be identified as mathematical expressions which must not be contracted and which must be punctuated appropriately.
3. TO, INTO, BY, and ARC. The disposition of these words cannot be determined until the succeeding word is scanned. If the second word is mathematical, the subroutine will precede the pair of words with a signal ($IG1) to the Translator that no contractions may be used.

The words in this table are located by means of a sub-table called the Pointer Table. A pointer entry for a given word is stored in the location (Pointer Table Origin - N) where N is the number produced by performing an EXCLUSIVE OR of all the characters in that word. This entry contains an index to the Word Table entry which contains the characters of the word. The adjacent entry in the Word Table identifies the subroutine which handles the word. Since the number of special words is less than 63, not all words in the Pointer Table will reference the Word Table. (Should future additions to the Table exceed this number, a seven-bit N can be produced by shifting certain bits before performing the OR.)

In some cases, two or more special words may produce the same N. For this reason, the Pointer Table entry contains a multiple entry code and the index to the first of these entries in the Word Table.

The look-up procedure is as follows:

1. When a word is identified as 'all alphabetic' or when the alphabetic characters of a punctuated word have been isolated, the characters of that input word are OR-ed. The resultant number is used to locate the Pointer Table entry.

2. If the Pointer Table word is zero, the input word is identified as non-special and is passed directly to the Translator.

3. If the Pointer word contains an index, the Word Table entry is compared, character-for-character with the input word. If there is a match, the appropriate subroutine is entered by referring to the adjacent Table entry.

4. If there is no match, and the Pointer word indicates that there is only one entry, the input word is passed directly to the Translator.

5. If there are multiple entries, the succeeding table words are compared with the input word. Since these entries are arranged alphabetically, additional entries will be examined only if the initial character of the input word is higher than the last table word examined. If it is lower, there is no possibility of a match among the other entries.

FORM FIRST LINE GRID

```
                 |---------------------|
                 |   PLACE ROW LABEL   |
                 |       IN GRID       |
                 |---------------------|
                            .
      (D)........V          V
                 |---------------------|
                 |  PLACE COLUMN LABEL |
                 |       IN GRID       |
                 |---------------------|
                            .
            .....V          V
            .    |---------------------|
      (E)...     |  FETCH INPUT CHAR.  |
                 |---------------------|
            .               .
            .               V
            .        XXXXXXXXXXXXX
            .       X             X
            .      X  UNDERLINE ?  X..(Y).......
            .       X             X            V
            .        XXXXXXXXXXXXX            (B)
            .               .
            .              (N)
            .               .
            .               V
            .        XXXXXXXXXXXXX
            .       X             X
            .      X    COMMA ?    X..(Y)......
            .       X             X            V
            .        XXXXXXXXXXXXX            (C)
            .               .
            .              (N)
            .               .
            .               V
            .       |-------------|
            .       |   STEP X*   |
            .       |-------------|
            .               .
            .................
```

*X - ITEM LENGTH

XMAXC - LONGEST ITEM IN
GIVEN EXAMPLE

E. TYPICAL SUBROUTINES

In the course of processing keypunched input for the Translator, it is frequently necessary to store a stream of characters and then to retrieve them for purposes of conversion or transmission. The routines which accomplish this are used not only by the main program, but also by other subroutines. Therefore, they are described here first.

Stack Routines

A stack is a method of storing information in a list using a completely self-contained subroutine. The unique quality of a stack derives from the routines to store and retrieve information from the list. These routines are completely self-contained and the user need not concern himself with the manipulation of index registers or the actual location of the stored data.

Essentially, these routines perform the functions outlined below:

```
Pointer = Pointer ± 1
Item ⇄ Stack

Save Pointer
Return
```

A frequent variety of stack uses the 'last-in, first-out' principle. Information is stored sequentially in the stack by placing it in the AC and calling a subroutine (DOWN). DOWN maintains a pointer which shows the last location in which significant information has been placed. To retrive information, a call is made to another routine (UP) which places the word in the AC and modifies the pointer to indicate that the next to the last word is now the last word.

Another version of a stack uses the 'first-in, first-out' principle. The DOWN subroutine works as above, but the UP routine retrieves information in the order in which it was placed in the stack.

These routines are extremely flexible. Switches indicating whether the stack is empty or full may be set, and in general, any condition of interest can be noted.

Format Routines

Although there will be many Format routines - one for each of the Format statements - they have in common the function of inserting spaces and line-ending codes to produce the desired alignment of characters. While some other examples are simpler, the XFORM routine is described here to illustrate the problems involved and the techniques used to solve them.

The XFORM routine is designed to arrange an array of exercises and to supply the appropriate labels for each example. An exercise array is a set of problems arranged for computation. No answers are shown. One such array, as it would appear in inkprint, is shown below.

```
                  a          b          c

     1.           2          5         13
                 +1         +6         +4
               ______     ______     ______

     2.           4         17          9
                 -2         -5         -6
               ______     ______     ______
```

The Braille rules require that the column identification (in this case, a, b, or c) be written to the left of each example, that a period and a space follow the letter, and that the examples be separated by two spaces. If a given row cannot be accommodated on one Braille line, it must be rearranged appropriately. While this is a time consuming task for the editor and the Braillist, it is essentially a matter of counting - a problem ideally suited to computer handling.

The keypunch instructions for this and other spatial arrangements are designed to minimize the decision-making and keystrokes required of the operator. (See KEYPUNCH) For this particular example, the operator punches an XFORM statement describing the array:

```
XFORM 2 rows (1. - 2.) 3 cols (a - c)
```

Following this statement, the problem material is punched as shown below:

```
2,+1// 5,+6// 13,+4// 4,-2// 17,-5// 9,-6//
```

The comma is used to separate one number (or operand) from another. The symbol // indicates an underline. Because the XFORM statement describes the row and column arrangement and labels, the operator can punch the exercises sequentially without indicating line endings. The XFORM routine will arrange the material and supply the labels, adding periods where necessary.

As indicated in the accompanying flow charts, there are two phases within the XFORM routine. The first of these is the development of a grid, or mask, containing the necessary spaces and labels. The second is the insertion of the exercise material in the appropriate locations.

A grid is a stack which represents the arrangement required for a given row in an array. Since it is necessary to determine the maximum width of a column, the grid for the first line of the row can only be formed after scanning all items in the row. The grid for all succeeding lines is a modification of the first one. To illustrate, the grids for the sample exercise array would be the following:

(* denotes a space)

```
First line:     1.*a.*00**b.*00**c.*00$EL*
Other lines:    ******00*****00*****00$EL*
```

The zeros indicate where the exercise material may be inserted in order that the columns be properly separated and aligned, and the number of zeros represents the size of the largest item in each example.

During the insertion, or Fill, phase, the exercise material is scanned backwards, one line at a time. The units position number is inserted in the grid first, then the tens position, and so on. If a particular item is smaller than the space allowed, the routine inserts a space code for each remaining zero in the grid. To accomplish this, a grid stack item and an input stack item are examined to determine what code should be placed in the output stack. The table below indicates the output code for each possible condition.

	Grid = 0	Grid ≠ 0
Input = number	number	grid item
= operator	operator	grid item
= comma	space	grid item
= underline	space	grid item

Selecting the input stack item to be examined involves skipping through the stack to pick out the factor for a given line. As each input item is moved to the output stack, a record is kept which causes the routine to ignore that item during subsequent scans.

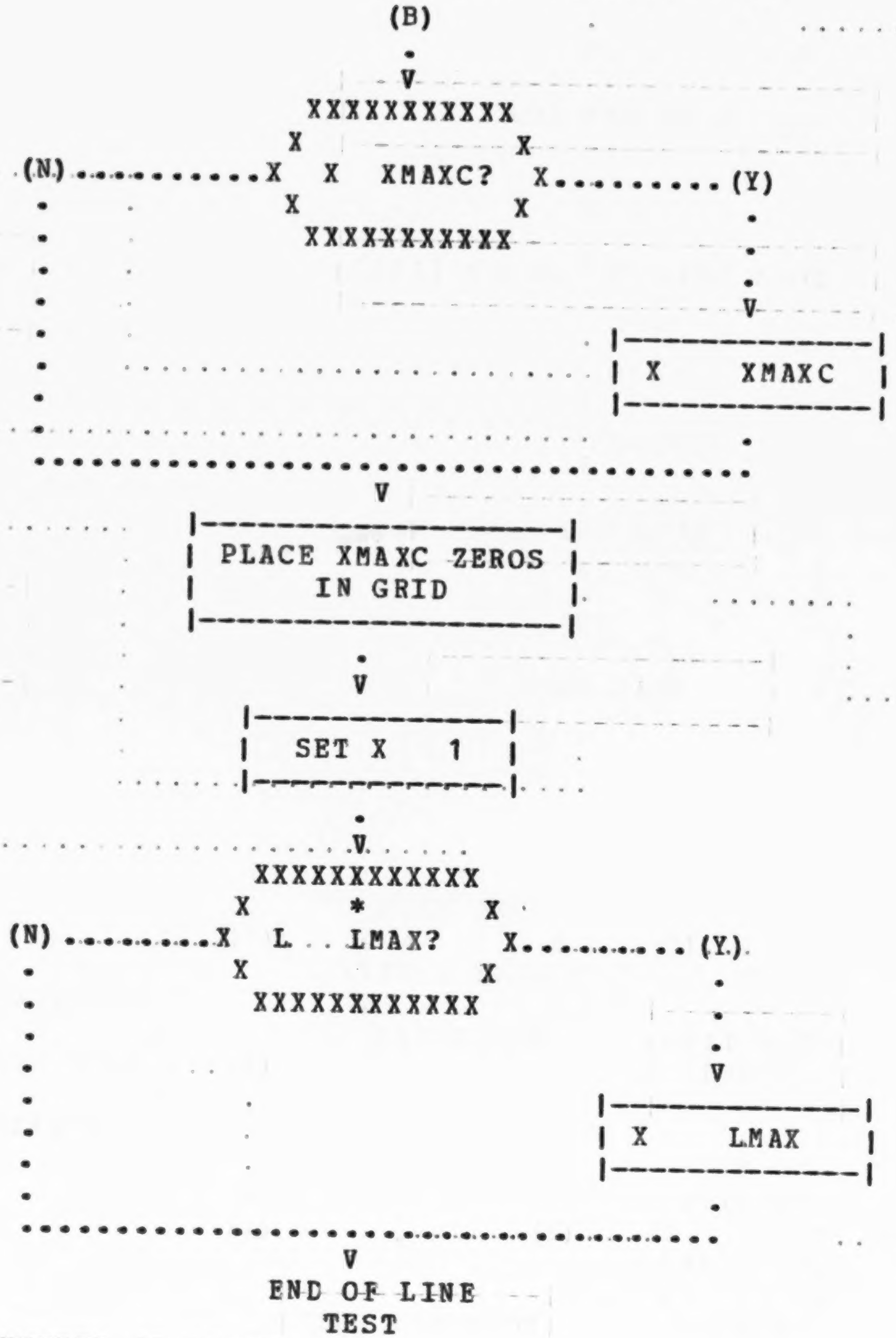
One important by-product of the grid making phase is a count of the number of lines required for a given row of the array. In general, formatting the Braille page is handled by the Translator. When a Format routine is entered, the usual word for word transmission is interrupted, and the Format routine reads and arranges a number of lines. These lines must be preceded by a signal to the Translator indicating how many lines there are. This signal ($ARAn) is used by the Translator to calculate whether the array can fit on the page currently being assembled, or whether a new page must be started.

Not indicated in the general flow chart is the fact that this routine supplies the appropriate pseudo-codes for letters and numbers which should not be preceded by indicators.

XFORM ROUTINE
GENERAL FLOW CHART

```
            |--------------------|
            |SAVE XFORM STATEMENT|
            |--------------------|
...........           .
.          V          V
.      |---------------------------|
(A)    |       SCAN ONE ROW        |
       |---------------------------|
                      .                       ...........
                      V                       .         V
       |---------------------------|          .     |--------|
       | FORM GRID FOR FIRST LINE  |          .     | OUTPUT |
       |---------------------------|          .     | $ARAn  |
                      .                       .     |--------|
                      .........................         .
                                                        .
                      ...................................
                      V
             |-----------------|
             |  SCAN ONE LINE  |              ...........
             |-----------------|              .         V
        ........      .                       .     |--------|
        .      V      V                       .     | OUTPUT |
        .   |-----------------|               .     | 1 LINE |
.........   |    FILL GRID    |               .     |--------|
.           |-----------------|               .         .
.                     .                       .         .
.                     .........................         .
.                                                       .
.                           .............................
.                           V
.                       XXXXXXXXX
.           ...(N).....X         X....(Y).........
.           V         X LAST LINE?X              V
.      |---------|     X         X           XXXXXXXX
.      |STEP LINE|      XXXXXXXXX           X        X
.      |  COUNT  |                    (N)..X LAST ROW?X..(Y)
.      |---------|                     .    X        X     .
.           .                          .     XXXXXXXX      .
.           V                          .                   .
.        XXXXXXX                       V                   V
 ..(N)..X       X.....(Y)........     (A)               RETURN
.      X2ND LINE?X            V
.       X       X      |-------------|
.        XXXXXXX       |FORM NEW GRID|
.                      |-------------|
.                             .
...............................
```

UNDERLINE


(B)
X XMAXC?
(N)
(Y)
X XMAXC
PLACE XMAXC ZEROS IN GRID
SET X 1
*
L LMAX?
(N)
(Y)
X LMAX
END OF LINE TEST

* L - NUMBER OF LINES
FOR GIVEN EXAMPLE

LMAX - NUMBER OF LINES, LONGEST EXAMPLE

END OF LINE TEST

```
                    |--------------------|
                    |                  * |
                    | BMAX - GCOUNT = Z  |
                    |--------------------|
                               .
                               V
                          XXXXXXXXXXX
                         X           X
           .....(N)......X  Z < 0?    X......(Y).....
            V            X           X              V
      XXXXXXXXXXX             XXXXXXXXXXX           TRUNCATE LINE
     X           X
(N).X  0≤Z≤4?     X.....(Y)......
 .   X           X              .
 .    XXXXXXXXXXX               V
 .                          END OF LINE
 .
 .
 ..........
           V
      XXXXXXXXXXX
     X           X
(N).X    LAST     X.....(Y).... .
 .   X COLUMN?   X              .
 .    XXXXXXXXXXX               V
 .                          END OF LINE
 .
(D)
```

* BMAX - MAXIMUM LENGTH BRAILLE LINE

GCOUNT - CURRENT GRID LENGTH

COMMA

```
                           (C)
                            .
                            .
                            V
                       XXXXXXXXXXX
                      X           X
 (N).........X X > XMAXC?  X..........(Y)
  .           X           X            .
  .            XXXXXXXXXXX             .
  .                                    .
  .                                    .
  .                                    V
  .                             |----------------|
  .                             | X  ->  XMAXC   |
  .                             |----------------|
  .                                    .
  ......................................
                            .
                            V
                     |-------------|
                     |SET X = 1    |
                     |-------------|
                            .
                            V
                   |-----------------|
                   | STEP LINE COUNT |
                   |-----------------|
                            .
                            .
                            V
                           (E)
```

FETCH
INPUT CHARACTER

```
                        FILL ROUTINE
........................V          V........................
.                      |------------|                       .
.                      | FETCH GRID |                       .
.                      | CHARACTER  |                       .
.                      |------------|                       .
.                                                           .
.                           V                               .
.                         XXXXXXXX                          .
.                        X         X                        .
.           ......(N)..X  = 0 ?     X..(Y)......            .
.           .            X         X          .             .
.           V     V...... XXXXXXXX            V             .
.      |-----------|                   |--------------|     .
.      | MOVE GRID |    .              | FETCH INPUT  |     .
.      | CHAR. TO  |    .              |  CHARACTER   |     .
.      |  OUTPUT   |    .              |--------------|     .
.      |-----------|    .                                   .
.           .           .                     .             .
.           .           .                     V             .
.           V           .                 XXXXXXXX          .
.                       .                X COMMA ?X         .
.      XXXXXXXXX        .       (Y).......X  UNDER-   X.(N) .
     X          X       .        .       X LINE ? X      .  .
.(N) .X LAST GRID X.(Y) .        V   V...  XXXXXXXX      .  .
     X CHAR. ? X        .  |-----------| .               V  .
      XXXXXXXXX         .  |   MOVE    | . |------------|   .
                        .  | SPACE TO  | . |    MOVE    |   .
                        .  |  OUTPUT   | . | INPUT CHAR |   .
                   V    .  |-----------| . | TO OUTPUT  |   .
              PUT OUT   .        .       . |------------|   .
               LINE     .        V       .                  .
                        . |-------------|.        ...........
                        . | FETCH GRID  |
                        . | CHARACTER   |
                        .       A
                        . |-------------|.
                        .      XXXX
                        .     X    X
                        . (N) X= 0 ? X(Y)..
                        .     X    X
                        .      XXXX
                        . V
                        . |-------------|
                        . | SKIP INPUT  |
                        . | POINTER TO  |
                        . | NEXT ITEM,  |
                        . | THIS LINE   |
                        . |-------------|
                        ..
                        ....
```

Math Scan:

The general flow chart in Section C refers to the M-Scan routine which controls the insertion of indicators and the selection of pseudo-codes when necessary. As shown in the more detailed flow chart in this section, M-Scan is a control routine which calls the highly specialized subroutines associated with each character in the Symbol Table. (See Section D.) Depending on the character being analysed, the individual subroutine may call for additional input characters, and may output one or more characters. The stack routines described earlier are particularly useful here. Both UP and DOWN routines will be called by many different subroutines; the stack routines themselves keep track of the pointers.

The functions of M-Scan can best be illustrated by tracing through the processing of the sample character string shown below:

(4,325)

1. The Left Parenthesis routine tests the Math Switch. Since it is on, the input character is modified by OR-ing it with a 100 code and the resultant pseudo-code is placed in the output stack. This routine also sets a switch which will be turned off by the Right Parenthesis routine.

2. The Math Switch setting causes the Number routine to convert the input code 4 to 104, which will be translated into the lower cell Braille code. The Translator will handle the insertion of the numeric indicator.

3. The Comma routine, after testing the Math Switch, will output the code for the mathematical comma.

4. The codes 3, 2, and 5 will be converted by the Number routine as described in 2. above.

5. The Right Parenthesis also is converted to the code which will produce the Braille code for the mathematical symbol. It also turns off the Grouping Symbol Switch.

6. The Space routine places the Space code, unchanged, in the output stack. Before returning to M-Scan, it tests the Grouping Symbol Switch. If it is on it means that there is more than one word enclosed and this routine would fetch additional input words until the closing symbol is read.

In this example, each routine produces output based on the setting of the Math Switch, without reference to adjacent characters. In some cases however, it is necessary to examine a preceding character before the proper output code can be selected. For example, a series of punctuation codes (.") requires the punctuation indicator only before the first one. Thus, the Period routine would place the Punctuation Indicator code in the output stack, followed by the Period. The Quote routine, after testing the preceding character, would output only the Quote code. To facilitate this testing, the indicator bits describing each character are held until the succeeding character is converted.

Because of the structure of the Symbol Table, M-Scan is a very flexible routine. Should future rules revisions require modifications in the handling of a given character, a new character subroutine can be easily introduced in this structure.

M - SCAN

```
                         |-------------------------|
                         | PLACE SPACE INDICATORS  |
                         |        IN PREV.         |
                         |-------------------------|
                                      .
        ...............V              V
        .                |-------------------------|
       (A)               |     FETCH INPUT CHAR.   |
                         |-------------------------|
                                      .
                                      V
                         |-------------------------|
                         | FETCH SYMBOL TABLE WORD |
                         |-------------------------|
                                      .
                                      V
                         |-------------------------|
                         |PLACE INDICATORS IN THIS |
                         |-------------------------|
                                      .
                                      V
                         |-------------------------|
                         |    ENTER  SUBROUTINE    |
                         |-------------------------|
                                      .
                                      V
           ...........................................
           V       V       V          V       V      V
         SPACE  NUMBER   ALPHA      COMMA  PERIOD   ETC
           .                                         .
           ...........................................
                                      .
                                      .
                                      .
                                      V
                                 XXXXXXXXXXXX
                   (Y) ......X      LAST     X..... (N)
                    .        X  CHARACTER?   X       .
                    .            XXXXXXXXXXXX        .
                    .                                .
                    V                                V
           |-----------|                       |-------------|
           |  OUTPUT   |                       |THIS -> PREV.|
           |-----------|                       |-------------|
                                                     .
                                                     .
                                                     V
                                                    (A)
```

F. KEYPUNCH INSTRUCTIONS:

Format statement:

Preceding each exercise, the operator must punch a statement indicating the format of the exercise array. (Array means the order and arrangement of the separate problems which make up the exercise.) For example, the array shown below has two rows, labelled 1. and 2., and three columns, labelled a, b and c.

$$\begin{array}{cccc} & a & b & c \\ 1. & \begin{array}{r}2\\ \underline{+1}\end{array} & \begin{array}{r}5\\ \underline{+6}\end{array} & \begin{array}{r}13\\ \underline{+4}\end{array} \\ 2. & \begin{array}{r}4\\ \underline{-2}\end{array} & \begin{array}{r}17\\ \underline{-5}\end{array} & \begin{array}{r}9\\ \underline{-6}\end{array} \end{array}$$

The format statement must begin with the key word XFORM, and end with ENDXR. It must indicate that there are 'n' rows (labels) and 'n' columns (labels). 'n' is the exact number of rows, or columns, and the labels must be punched as they appear in the inkprint text. Where there are more than two rows or columns, only the first and last label should be indicated. The format statement for the example shown above should be as follows:

```
XFORM 2 rows (1. - 2.) 3 cols (a - c) ENDXR
```

Each format statement should begin in column 1 of a new card. The problem material should also begin on a new card.

Problems:

The problems making up the exercise should be punched row for row. Do not punch labels. Beginning in column 1, the exercise shown above should be punched as follows:

```
2,+1// 5,+6// 13,+4// 4,-2// 17,-5// 9,-6//
```

The comma is used to separate one number (or operand) from another. The following problem:

$$\begin{array}{r}15\\ 11\\ \underline{+3}\end{array}$$

should be punched:

```
15,11,+3//
```

Exercise material should be punched continously, using 72 columns of each card. If fewer than 72 columns are used, no end-of-card indicator is required.

Multiplication:

Illustrated below are what we define as 'standard' worked-out multiplication formats and the keypunching instructions for handling them.

Definition of terms:

```
Multiplicand
  Multiplier
 ___________
 Part. Prod.
Part. Prod.
 ___________
     Product
```

Notation: x represents any digit 1-9; o represents zero.

Examples:

```
(1)      xxx             (2)      xxx
          xx                      xox
         ---                      ---
         xxx                      xxx
        xxx                      xxxo
        ----                     -----
        xxxx                     xxxxx
```

Note:
1. The multiplicand, multiplier and product are right justified.
2. The number of non-zero digits in the multiplier is equal to the number of partial products.
3. The first partial product is right justified with the multiplicand, multiplier, etc. Succeeding partial products are shifted left one position each.

Punching instructions:

Precede each example with $WOX * and follow each with $EWOX. Punch the example line for line. If the multiply sign is shown in inkprint punch *. (This option is illustrated below as (*).)

Multiplicand, (*)Multiplier//Partial product,partial product//Product

(1) xxx,xx//xxx,xxx//xxxx

(2) xxx,xox//xxx,xxxo//xxxxx

The following variation may be punched following the same punching instructions:

```
(3)     xxx
        xox
        ---
        xxx
      xxx
      -----
      xxxxx
```

In this case, the zero multiplier digits produces a space (not a zero) in the second partial product. The operator must include this space.

(3) xxx,xox//xxx,xxxspace//xxxxx

Division:

Illustrated below are what we define as 'standard' worked-out division formats, and the keypunching instructions for handling them.

Definition of terms:

```
          Quotient
         ----------
  Divisor ) Dividend
             P.P.       (partial product)
             ----
               P.P.
               ----
               Remainder
```

Notation: x represents any digit 1-9; o represents zero

Examples:

```
(1)        xxx              (2)         xox
        -------                      -------
      xx)xxxx                     xx)xxxx
         xx                          xx
         --                          --
          xx                          xxx
          xx                          xxx
          --                          ---
           xx                          xx
           xx
           --
            x
```

Note:
1. The number of non-zero digits in the quotient is equal to the number of underlines.
2. The quotient, dividend and remainder (if shown) are right justified. (i.e. the rightmost digit of each is on the same vertical line.)
3. Each non-zero quotient digit is right justified with a partial product.

Punching instructions:

Precede each worked-out example with $WOX)- and follow each with $EWOX. Punch the example itself line for line as follows:

> Quotient, Divisor)-Dividend,P.P.//P.D.,P.P.//P.D., P.P.//Remainder
>
> (P.D. is that part of the dividend which is 'brought down'; P.P. is the partial product - one quotient digit times the divisor.)

(1) xxx,xx)-xxxx,xx//xx,xx//xx,xx//x

(2) xox,xx)-xxxx,xx//xxx,xxx//xx

The following examples represent minor variations from the 'standard' and can be handled using essentially the same keypunching instructions:

```
(3)           xx Rx
          ____
      xxx)xxxx
          xxx
          ---
           xxx
           xxx
           ---
             x
```

In this case, the remainder is shown on a line with the quotient. It may or may not be shown as the end of the example. Again, this should be punched line for line, leaving a space between the quotient and the remainder.

(3) xx Rx,xxx)-xxxx,xxx//xxx,xxx//x

```
(4)          x
         ____
      xx)xxxx
         xx
         --
          xx
```

The example above is only partially worked-out, and the operator must indicate the blank positions in the quotient, but otherwise punch line for line as shown in inkprint.

(4) xspacespace,xx)-xxxx,xx//xx

In figuring the number of spaces, remember that the fully worked out quotient is right justified with the dividend. The number of spaces is the number of dividend digits to the right of the partial quotient.

Keypunching of Worked Out Examples

Worked out examples are those in which the actual process of computation is shown. Thus,

$$\text{a.}\quad \begin{array}{r} 1 \\ 25 \\ \underline{+7} \\ 32 \end{array} \qquad \text{b.}\quad \begin{array}{r} 23 \\ \underline{\text{x}54} \\ 92 \\ \underline{115} \\ 1242 \end{array} \qquad \text{c.}\quad \begin{array}{r} 52 \\ 23\overline{)1196} \\ \underline{115} \\ 46 \\ \underline{46} \\ 0 \end{array}$$

are worked out examples, but

$$\text{d.}\quad \begin{array}{r} 25 \\ \underline{+7} \\ 32 \end{array} \qquad \text{e.}\quad \begin{array}{r} 23 \\ \underline{\text{x}54} \\ 1242 \end{array} \qquad \text{f.}\quad \begin{array}{r} 52 \\ 23\overline{)1196} \end{array}$$

are not. This is because (a) shows that a carry has been taken into the ten's column, while (d) does not; (b) shows the formation of the partial products (92 and 115), while (e) does not; and (c) shows the multiplications and subtractions necessary to perform long division, while (f) does not.

In keypunching worked out examples, four things are necessary:

(1) a statement that what follows is the description of a worked out example; this is indicated by punching $$WOX;

(2) a statement of the format of the worked out example;

(3) the listing of the characters, spaces, and special symbols which make up the body of the worked out example;

and (4) a statement signifying the end of the worked out example; this is indicated by punching $$EWOX.

Because of the fact that worked out examples usually have odd spatial arrangements, items (2) and (3) are handled by reading the worked out example vertically instead of horizontally. For example, in (b), part of the list of characters forming the worked out example would read, when punched,

$$3,4,2,\Delta,2;$$

note that this list of characters is simply the rightmost column of numbers and spaces appearing in the example, read from top to bottom, excluding underlines. The underlines are not shown because the format statement will show where they occur.

The following is a description of how to punch a worked out example.

(1) When a worked out example is encountered, punch the example number, etc., followed by $$WOX.

(2) Specify the format. In order to do this, it is suggested that some sort of straightedge, for instance a three by five index card, be placed on the page to the right of the example so that a vertical edge runs parallel to the rightmost part of the example. Then pick out the longest vertical column of symbols in the example; that is, the one which includes the top horizontal line of characters, symbols, etc., in the example and which also includes the bottom horizontal line of characters, symbols, etc. in the example. Place the vertical edge of the card immediately to the right of this column. Punch a left parenthesis, followed by all the symbols which appear in the column from top to bottom, separated by commas, and followed by a right parenthesis. Note that from any horizontal line of the example, only one symbol may be selected; thus if '2.' appears in a horizontal line of the example, either '2' or '.' may be punched, but not both. All underlines are considered to be separate horizontal lines of the example, and must be included. For the given examples, the following would be correct format statements:

a) (1,2,Δ,_,3)

b) (3,4,_,2,Δ,_,2)
or (2,5,_,9,5,_,4)

c) (2,_,6,Δ,_,6,6,_,0)

Note that all underlines are included.

(3) Specify all the characters in the worked out example. To do this, place the vertical edge of the card to the immediate right of the rightmost column of the example. Punch a left parenthesis, and then the symbols as they appear, top to bottom, just as in the format statement, but omitting the underlines; at the end of the column punch a semicolon, and then proceed exactly the same with the next column of the example to the left, covering the column just punched with the card. Continue until through, and then punch a right parenthesis. Spaces at the tops of columns and within columns must be included, but the semicolon may be punched as soon as the last

character in the column is recorded.
(4) At the end of the example, punch $$EWOX.

When correctly punched, the examples given above, might appear thus:

a) $$WOX (1,2,Δ,_,3) (Δ,5,7,2;1,2,Δ,3;Δ,Δ,+;) $$EWOX (the final semicolon must be included; see below)

b) $$WOX (3,4,_,2,Δ,_,2) (3,4,2,Δ,2;2,5,9,5,4;
Δ,x,Δ,1,2;Δ,Δ,Δ,1,1;) $$EWOX

c) $$WOX (2,_,6,Δ,_,6,6,_,0) (2,6,Δ,6,6,0;5,9,5,
4,4;Δ,1,1;Δ,1,1;Δ,);Δ,3;Δ,2;)$$EWOX

Keypunching subscripts and superscripts.

A subscript is a character that is affixed to another character on either the left or right, but below the line of print of that character. Thus, in:

$$X_{a_2}$$

'a' is a subscript to 'x', and '2' is a subscript to 'a'.

A superscript is a character affixed to another character on either the left or right but above the line of print of that character. Thus in:

$$X^{a^2}$$

'a' is a superscript to 'x', and '2' is a superscript to 'a'.

Note that subscripts may have subscripts, and superscripts may have superscripts. It is also true that subscripts may have superscripts and that superscripts may have subscripts. It is always true that if a subscript has a subscript, then the subscript to the first subscript will be in a smaller type face than the first subscript. This applies also to the case of a superscript to a subscript, a superscript to a superscript, and a subscript to a superscript. Thus in

$$X_{2^a}$$

'2' is a subscript of 'x' and 'a' is a superscript of '2';

in

$${}_{2^a}X$$

'2' is a subscript of 'x', and 'a' is a superscript of '2'.

Thus, in any case where there is a possibility of confusion as to what is a superscript and what is a subscript, the relative sizes of the type faces will solve the problem. Such cases are encountered very rarely.

In all the examples given so far, 'x' is said to be on the

Base Level; that is the main line of printing when all sub- or super- scripts are disregarded.

In keypunching superscripts and subscripts, we always begin relative to the Base Level, and then move up or down from it using the level operators $UP and $DN. That is, if we punch a symbol which is on the Base Level, we may punch a superscript to it by punching $UP and then the superscript; if we wish to punch a subscript to the original symbol, we would punch the original symbol, $DN and then the subscript. If we wished to punch a superscript to a superscript to a symbol, we would punch the symbol, $UP, the first superscript, $UP again, and then the superscript to the superscript. If we wished to punch a symbol which had a superscript, and a subscript to the superscript, we would punch the symbol, $UP, the superscript, $DN, and then the subscript to the superscript. When we have finished punching the various superscripts and subscripts, we return to the Base Level of printing by punching $BL.

Here are some examples.

1. for: $\log_{10}$

   punch: log $DN 10 $BL

2. for: $ax^{2} + by$

   punch: ax $UP 2 $BL +by

3. for: $x_{1} + y_{2}$

   punch: x $DN 1 $BL + y $DN 2 $BL

4. for: $x^{a+b} + c$

   punch x $UP a+b $BL + c

5. for: $x^{a^{2}}$

   punch: x $UP a $UP 2 $BL

6. for: $x^{a_{2}}$

   punch: x $UP a $DN 2 $BL

7. for: ${}_{a^{2}}X$

punch: $DN a $UP 2 $BL x
here, note that the difference in type size showed that '2' was the superscript to the subscript 'a'.

8. for: ${}_{a_{2}}X$

punch: $DN a $DN 2 $BL x

In these examples, the blanks are included only for clairty.

Simultaneous subscripts and superscripts:

Sometimes we have an expression such as:

$$X_{1}^{2}$$

that is, 'x' simultaneously has a superscript and a subscript. For this we use the backspace operator, $BS. For instance, for the example above, we would punch first,

x $UP 2 $BL,

which means x^{2}; we then use $BS which returns us to the symbol 'x', and then punch the subscripts, giving

x $UP 2 $BL $BS $DN 1 $BL

which means: X_{1}^{2}

Other examples are:

1. for:

$$X_{c+d}^{a+b}$$

punch: x $UP a+b $BL $BS $DN c+d $BL

2.
for: $X_1^{a^2}$

punch: `x $UP a $UP 2 $BL $BS $DN 1 $BL`

3.
but for: $X^{a^2}{}_{1}$

punch: `x $UP a $UP 2 $BL $DN 1 $BL`

Complex subscripts and superscripts:

Some times we encounter expressions such as this:

$$X^{a^3+b^2+c_1}$$

that is, where the superscript (or subscript) is a complex expression in itself, involving subscripts and superscripts to the original superscripts (or subscripts) plus other symbols on the same level as the original superscript (or subscript), but not on the Base Level. In order to be able to punch this, we would like to establish the level of the original superscript as a sort of temporary base level, which we shall call the Current Level.

In order to do this we introduce three operators, $BCL, Begin Current Level, $ECL, End Current Level, and $CL, Return to Current Level. When a Current Level has been established by the use of $BCL, any use of $CL will act like $BL, except that the level returned to will be the one established by the use of $BCL. The use of $ECL will cancel the effect of the first previous $BCL.

As an illustration, suppose we wish to punch the expression above,

$$X^{a^3+b^2+c_1}$$

In order to do this, we do the following.

First, we punch: `x $UP $BCL`

This says that x on the Base Level has a superscript, and that we wish to establish the superscript level as a temporary base level, or Current Level; we then punch the first term according to the usual rules, giving:

```
x $UP $BCL  a $UP 3
```

this means we have, so far,

$$X^{a^3}$$

We now punch $CL, giving us

```
x $UP $BCL a $UP 3    $CL
```

this returns us to the level of the first 'a', the level we established as the Current Level by using $BCL. We now punch

```
+b $UP 2
```

giving us

```
x $UP $BCL a $UP 3 $CL   +b $UP 2
```

which means

$$X^{a^3+b^2}$$

We are now at the level of the superscript, '2' of 'b', and return to the level of 'a', '+', and 'b' by using $CL, giving

```
x $UP $BCL a $UP 3 $CL +b $UP 2 $CL.
```

Similarly we punch the last term (which involves a subscript) as +c $DN 1, giving

```
x $UP $BCL a $UP 3 $CL +b $UP 2 $CL +c $DN 1
```

which means

$$X^{a^3+b^2+c_1}$$

We return to the Current Level by using $ECL which tells us to go back to the Current Level and also that we are through with the Current Level:

```
x $UP $BCL a $UP 3 $CL +b $UP 2 $CL +c $DN 1 $ECL,
```

and finally we return to the Base Level by using $BL, giving us:

```
x $UP $BCL a $UP 3 $CL +b $UP 2 $CL +c $DN 1 $ECL
$BL.
```

Another example is this: we wish to punch

$$x^{a^{b^2+c}+g}$$

We punch:

```
 x $UP $BCL a $UP $BCL b $UP 2 $CL +c $ECL +g $ECL $BL
 1  2    3  4  5     6  7  8  9  10 11  12  13  14   15
```

We will go through this step by step (as numbered) showing what happens.

1. This gives us x.
2. $UP notifies us that what follows is a superscript.
3. $BCL notifies us that we are temporarily establishing a Current Level as the superscript to 'x'.
4. 'a' says that the superscript starts with 'a', that is, we now have

$$x^{a}$$

5. $UP says that we have a superscript to the 'a'.
6. $BCL says that we are establishing another, new temporary Current Level as a superscript to 'a'.
7. 'b' says that the superscript to 'a' starts with 'b'; we now have:

$$x^{a^{b}}$$

8. $UP says that 'b' has a superscript.
9. '2' gives this superscript; we now have:

$$x^{a^{b^2}}$$

10. $CL says that we must return to the last established Current Level; that is the level of 'b'.

11. '+c' gives us;

$$x^{ab^2+c}$$

12. $ECL says that we are through with the last established Current Level and are to return to the last previously established Current Level, namely, the one for the original superscript 'a'.
13. '+g' gives us +g on the level of 'a':

$$x^{ab^2+c+g}$$

14. $ECL says that we are through with the Current Level associated with 'a'.
15. $BL returns us to the level of 'x'; that is the Base Level.

One more example:

for:

$$x^{ab^2+c}$$

we would punch:

```
x $UP a $UP $BCL b $UP 2 $CL +c $ECL $BL
```

we could also punch:

```
x $UP a $UP $BCL b $UP 2 $CL +c $BL
```

That is, the use of $BL returns us to the Base Level, no matter how many temporary Current Levels we have established using $BCL.

To summarize:

We have introduced several control operators: $UP, $DN, $BL, $BS, $BCL, $CL, and $ECL.

Use of $UP causes all succeeding characters to be treated as superscripts unless another control operator is encountered.

Use of $DN is the same as $UP, except that subscripts are produced.

Use of $BL causes all succeeding characters to be treated as if they were on the main, or Base Level of printing.

Use of $BS allows simultaneous subscripts and superscript to a symbol; when the sub- and super- scripts to a single symbol are printed one above another in text, $BS must be used.

When we encounter a complex sub- or super- script, that is, one in which we must be able to return not only to the Base Level, but also to the level of other sub- or super-scripts, we may temporarily establish a Current Level using $BCL.

After using $BCL, the use of $CL returns us to the level established by the last $BCL used.

Use of $ECL cancels the effect of the last $BCL issued.

$BL always cancels the effect of all previous $ECL operators, and returns us to the main line of printing.

G. SUMMARY

A design has been described to implement the translation of mixed literary and mathematical materials into the appropriate codes. The program plan uses, in part, the literary translation program currently in operation at the American Printing House for the Blind with the fewest possible changes.

While the Nemeth Code is currently undergoing revision, the scheme described is sufficiently flexible to handle any changes. From a computing viewpoint the changes, in general, will make the programming task more straightforward.

Prepared by Ann and Joseph Schack, Schack Associates,
June - July 1968

Appendix B

A COMPUTER PROGRAM TO PRODUCE MUSICAL BRAILLE: IS IT FEASIBLE?

Ann and Joseph Schack
Schack Associates

TABLE OF CONTENTS

I. INTRODUCTION.. 70
II. INPUT... 71
III. COMPUTABILITY... 73
IV. OUTPUT.. 76
V. SYSTEM DESIGN... 77
VI. SUMMARY... 78

I. INTRODUCTION

This report, based on a preliminary study of the problem, answers the title question affirmatively. A computer program to produce Braille music is feasible. The development of such a program to handle the variety of musical material is a lengthy and complex project. It is best accomplished in a series of steps analogous to the phases in the training of a Braillist. That is, the initial effort should be to develop a program to translate monophonic music. After a suitable testing period, this basic structure should be elaborated to handle more complex material.

In determining whether any human activity can be handled by a computer program, three different areas must be examined:

1. Input: Can the source data be transcribed into machine readable form -- or is it already available in such form?

2. Computability: Are there rules which govern the function to be performed, and can these rules be codified for computer handling?

3. Output: Can the computer produce output in a usable form?

The answers to these questions are detailed in this report. They form the basis for the authors' judgment that a Musical Braille Translation program is feasible.

II. INPUT

The preparation of input for the Literary Braille Translation program is fairly straightforward. Although some special codes were required to describe format conditions, the basic character set is that which can be produced using a standard keypunch - the alphabet, numbers and punctuation marks. This is not true of the character set used in inkprint music. The 'alphabet' of musical notes is quite extensive. The significance of a note is indicated by its shape and color, its location on a grid (stave) and the definition of that grid (G-clef, F-clef, etc.). The meaning of a given note may also be influenced by other symbols (accidentals, marks of articulation, ties, slurs, etc.) which may or may not be adjacent to the note itself. All of this information must be represented in digital form for computer handling.

It is a fortunate coincidence that there is a project underway to automate the setting of musical type. It is even more fortuntate that the authors were able to interest one of the principal investigators of that project in serving as consultant to the Musical Braille sutdy. Mr. Stefan Bauer-Mengelberg and his colleagues have spent nearly three years developing a digitalized representation of musical notation to serve as input to a musical typesetting program. The conditions which gave rise to both projects are identical: the need for good inkprint (and Braille) music; and, the difficulty of obtaining the services of skilled musical typesetters - or Braillists. The digitalized input language for typesetting has therefore been designed not only with a view toward its computer legibility, but also its unambiguity and the ease with which it can be learned and executed by a non-musician.

These are the same aims which should be met by an input language for a Braille Music program. After considerable study the authors conclude that this language is ideally suited as an input language for Braille Translation. In a sense, the use of this language is analogous to the use of the 'input' Teletypesetter tapes as input to the Literary Braille Translation program. (When automated music typesetting has become an established practice, it might be interesting to consider the use of the typesetting medium itself as input to the Braille program, as has been done with Literary Braille. This eventuality is too distant to merit more than a passing comment here.)

Mr. Bauer-Mengelberg is not only willing to make this language available, but is currently engaged in training us in its philosophy and the details of its application. In essence, the language requires the transcriber to define at least two characteristics of a given note: its duration (shape and color) and its location. A single alphabetic character describes duration, and wherever possible, it has mnemonic value. (E.g., W = Whole note, H = Half, etc.) The location of the note is indicated by a number which defines the line or space on which it lies. The numbering system is simple and the user need not know the musical significance of the note.

It is difficult to state how many characteristics might be defined at most. There are provisions in the language for accidentals, slurs, ties, stem position, articulation marks, etc., any number of which might influence the significance of a given note. It is important to emphasize that the transcriber need not know the musical significance of these signs but simply that a given symbol should be represented by a particular character or group of characters. In some cases, the inkprint symbol can be produced by the keypunch: the sharp is represented by the number sign (#); the dot following a note, by the period (.). In other cases, the assignment of symbols is arbitrary: the natural (♮) is punched *; the flat (♭), represented by the hyphen (-). The keypunch operator is not concerned with the fact that the sharp sign changes the sound of the note, or that the dotted eighth note is longer than an eighth, but only with the selection of the proper code.

Had this language not been developed, this report would have been addressed to that major problem. Because this digitalized system has been designed and is available to us, attention can be directed to the second question - that of computability.

III. COMPUTABILITY

Serious consideration of the rules of Musical Braille and consultations with Mr. Bauer-Mengelberg and Mrs. Nelle Edwards have convinced the authors that these rules can be codified for computer handling. The kind of computer processing required can best be illustrated by considering some of the rules which must be applied. For convenience here, the rules are divided into two categories - translation and format.

Translation rules

The translation process is essentially a process of substitution: a given input symbol (or set of symbols) is replaced with a given output symbol (or set of symbols). The human transcriber accomplishes this substitution by reference to memorized or documented tables. The computer is programmed to 'look-up' the symbol in a table or dictionary stored in the machine's storage unit. However, because a single note symbol frequently conveys more information than can be represented by a single Braille cell, the substitution performed is rarely one-for-one. For example, a single Braille cell describes the duration of a note and its position within one octave. The octave must be defined by another symbol, the effect of which continues until another octave code is encountered.

Another problem is raised by the fact that the Braille cell for a given note is multiply used. That is, the same cell represents a whole note and a sixteenth, a half note and a thirty-second, etc. The meaning is usually clear from the context, but where it is not, a value sign is introduced. Therefore, even the translation of a single note requires a scan of the preceeding conditions to determine whether an octave sign and/or a value sign is required.

A longer scan is required to apply the rules governing the grouping of notes (16ths, 32nds, etc.). The basic rule states that the first note of the group is written in its true value, the others, as eighth notes. The exceptions to this statement require that the program scan ahead, possibly to the end of the line, before the correct Braille can be output. For example, this method of grouping cannot be used if:

(a) the group is followed by an eighth or dotted eighth,
(b) the group contains a rest on any other note but the first,
(c) the group cannot be completed in the line in which it begins.

A scan of greater complexity is required in determining whether the repeat sign can be used. This Braille symbol represents a major difference between ink-print and Braille music, and is used to save space and to facilitate reading and memorizing. Because inkprint music is often 'sight-read' such a symbol would not be of similar value. It is not appropriate to detail here the many conditions which determine whether the repeat sign may be used. It is important to note that a repeat sign may be used to represent a part-measure or a whole measure. It may be used for a passage played in a different octave from the original, and it may be used, with a number, to indicate that a passage is repeated more than once. This means that the program must continually scan to determine whether a given set of notes is repeated, and then must test numerous other conditions which restrict the use of the repeat sign.

These few rules have been chosen to illustrate the nature of the processing to be performed by the computer. They represent only a small sub-set of the translation-type rules of the Music Code. These rules will be effected by a combination of table-search techniques and scanning techniques. As illustrated above, the scan may be of a single preceeding note, a whole line, or many lines.

Format rules

The arrangement of Braille music on a page is extremely important, and, of necessity, quite different from inkprint arrangement which is designed so that the eye can take in several staves at once. The Braille reader can read only one or two cells at a time but must still be able to determine that several notes are to be played simultaneously. To facilitate this, the Braille music is arranged in parallel lines as in inkprint. The computer processing required to apply the specific rules of format will range from a scan and count of notes within a single measure, to the tentative arrangement of an entire page. There are special format requirements determined by the type of music (e.g., vocal, conductor's score, etc.) which add to the complexity of the format processing.

A few rules are described here to illustrate the range of scanning and counting required. A format routine which will be frequently executed is necessitated by the requirement that the first note of every measure in keyboard music be given its appropriate octave mark in all parts. Another rule which requires the insertion of additional information for the Braille reader is that which states that measures be numbered at the beginning of every parallel.

One of the more complex problems is that of measure division. If the measure cannot be completed in all parts in the line in which it begins, it must be divided at the same point in every part. In complicated music, application of this rule will require extensive scanning and counting. Many lines of information must be saved and assembled before the end of line points can be determined.

Deciding where a page should end will also require saving many lines of material. One rule states that a parallel must always be completed on the page in which it begins. Since a parallel may be composed of many staves, applying this rule requires a lengthy scan.

It is clear that formatting Braille Music is even more formidable than arranging Literary Braille. These examples show some of the many points at which format decisions must be made.

IV. OUTPUT

The International Braille Music Code consists of the same six-dot cells as the Literary Braille Code. (Of course, the meanings are entirely different.) Since the Music Translation program will have to interface with the Literary Braille Translation program to produce lyrics and musical material embedded in otherwise literary texts, the output will be the same punched cards which operate the card-controlled stereograph equipment.

V. SYSTEM DESIGN

A primary consideration in the design and programming of Musical Braille is the need to expand from a system capable of handling relatively simple music to one which can translate the more complex forms. To accomplish this we envision a series of boxes, (independent programs which can be linked together), each of which performs one necessary function. A list of possible boxes and a brief description of each follows.

INBOX
SCANNER
MUTRAN
FORMAT
OUTBOX

Inbox

Inbox is the input interface with the literary program. It receives the key-punched information described in Section II.

Scanner

Scanner translates the keypunched input into the Universal Music Code (UMC).

Mutran

Mutran translates UMC into Braille and outputs a stream of Braille Music codes.

Format

Format handles the stream of Braille Music codes and by inserting the necessary pseudo-codes controls the formatting of the Braille music.

Outbox

Outbox transmits the Braille Music codes and pseudo-codes to the Literary program and returns to the Literary program when the current musical input has been translated.

VI. SUMMARY

The present study of Musical Braille Notation has led to the conclusion that an automated translation system can be implemented on a computer. Some of the problems have been discussed, as well as solutions, and some highly preliminary system design has been presented.

PART II

STUDIES TOWARD ADVANCING COMPUTER TRANSLATION OF ENGLISH BRAILLE

Robert Haynes and John Siems

SUMMARY

Since 1964 computer translation has formed a part of the Braille publishing process at the American Printing House for the Blind. After two years, experience in production provided a basis for the research covered in this report.

Because computer translation did not provide for hyphenation, a study of space requirements for hyphenated versus unhyphenated Braille was undertaken. The results indicated a savings of 0.7% when hyphenation was used. The small amount of space saved does not appear to justify a hyphenation program.

In 1968 a translation program incorporating new principles had been completed. This program used as the basic unit of translation the Braille signs. Also, a distinction is made between general rules and exceptions to the rules. After almost two years of production this program has proven effective in extending translation capabilities.

A study of computer translation results indicated a need for proofreading in order to meet publications standards. Also, the time at which proofreading is most effective was determined to be after plates have been embossed.

A cost analysis of Braille plates revealed that in the manual production system proofreading is the most expensive operation. Input preparation was clearly established as the most expensive operation in the production of computer Braille. All areas in computer translation have a potential for cost reduction from technological advances.

This report is accompanied by a document entitled "A Program Abstract for Translating Inkprint into Braille by Computer". The abstract shows the approach used in the 1968 Braille translation program written for the IBM 709.

A benchmark was written to determine the effectiveness of third generation computers in Braille translation. A computer having the capacity of the IBM 360/40 appears to be appropriate for Braille production.

A survey of contraction usage in various contexts of pronunciations and syllabification supplements this report.

INTRODUCTION

Computer processing began to be used regularly for Braille translation at the American Printing House for the Blind in 1964. By means of a Grade 2 Braille translation program written in the 1950's by International Business Machines in cooperation with the American Printing House for the Blind, an IBM 709 Data Processing System installed at the Printing House converted keypunched inkprint data into punched cards containing Braille text data and codes for format control. Automatic stereograph equipment, which had been developed at the American Printing House for the Blind, had the capability of reading these output cards and making metal plates for Braille embossing. This system provided a new means for producing certain types of Braille literature from inkprint and thus for meeting needs of blind readers. Previously, the only method available for making Braille plates was the manual operation of stereograph machines by Braille transcribers.

After a number of months of operation, possibilities of the automated system had been demonstrated. Also, areas for further research and development had been highlighted. Experience with problems encountered in regular production provided a basis for further exploration of Braille translation. Other research resources were the amassed inkprint and Braille data and the availability of the 709 computer.

A group of studies was undertaken directly related to the translation of the literary code of English Braille. These studies are reported here. Research into the translation of two other codes is reported in Part I.

One area of Braille translation requiring attention was the question of hyphenation. Programming for translation had not made provision for dividing words at the end of a line. It appeared that a study of the effect of hyphenation upon space requirements for Braille text would be of value.

During the first years of Braille translation by computer, revisions had been made in the original translation program in response to situations arising from the variety of styles used in literary texts. Additions were also made to handle more types of format. It seemed that it would be desirable to develop a new program extending computer translation to a wider range of materials and using an approach which would provide flexibility and increased accuracy in solving specific problems which had been encountered in production over a period of time.

Braille translation differs from most other computer applications in that the highly unpredictable nature of the input limits the use of validity checks. As a text processing operation the internal verification of results is also to a large extent precluded. There was a need for some research directed toward making the proofreading phase of the procedure as effective as possible.

Production experience provided an opportunity to analyze the various phases of the automated procedure with reference to cost factors to show the relative cost of each phase.

From the standpoint of efficiency of computer operation it seemed advantageous to write the new Braille translation program in assembler language. For the same reason, the original Grade 2 translation program had also been written in assembler language. However, in view of the interest of many organizations and institutions in Braille translation by computer, there is a need for a presentation, on a higher level, of logical concepts in computer translation. An abstract of the new program is directed toward this need.

It appeared that it would be of interest to investigate the potential of third generation computers for Braille translation. This suggested the development of simulation programs and a benchmark program.

The part of a Braille translation program which is most subject to change is that part which controls use and non-use of the Braille contractions in unusual words. Exceptions arise because of syllable division or other features of these words. It seemed appropriate to explore this area more fully.

This report presents results and developments from research in these areas of Braille translation by computer.

SPACE REQUIREMENTS FOR HYPHENATED VERSUS UNHYPHENATED BRAILLE

The computer translation program at the American Printing House for the Blind does not hyphenate words appearing at the ends of lines. This is the only point at which computer produced Braille differs from Braille produced by manual transcribers. Since this difference does exist a logical question is, How much space is saved by hyphenation?

An effort toward answering this question was made by comparing samples of Braille text translated by both automated and manual methods. The sample material consisted of the first 50 pages of each of ten books by different authors. These books had been transcribed into Braille by eight different Braillists operating stereograph machines and following standard hyphenating procedures. The same text was translated by computer without hyphenation.

Results were:

Source of Text Sample	Number of Pages	Lines in Hyphenated Copy	Lines in Unhyphenated Copy	Space Saved by Hyphenation
A STUDY OF COMMUNISM	45	1050	1058 1/4	0.79%
ENTER, CONVERSING	50	1227	1235	0.65%
MODERN AMERICANS IN SCIENCE AND TECHNOLOGY	50	1250	1255	0.40%
NEFERTITI	50	1250	1261	0.88%
THROUGH DARKEST ADOLESCENCE	50	1250	1272	1.76%
THE GREEN FAIRY BOOK	50	1250	1246	-0.32%
RAISE HIGH THE ROOF BEAM, CARPENTERS, etc.	50	1600	1615	0.94%
RETURN TO THE WILD	50	1250	1266	1.28%
MARKINGS	50	1242	1233	-0.72%
AFRICAN CREEKS I HAVE BEEN UP	50	1250	1267	1.36%
	495	12,619	12,708	0.70%

The average amount of space saved by hyphenation was .70%. The amounts for individual books varied from -.72% to +1.76%. In certain cases because of its effect upon contraction usage, hyphenating required more space than not hyphenating. Usually, however, the space required by hyphenated Braille was somewhat less. On the basis of the results obtained using this sample, it can be expected that 100 lines of hyphenated Braille will require 100.7 lines of unhyphenated Braille.

1968 BRAILLE TRANSLATION PROGRAM

In the 1950's a Braille translation program was written for the IBM 704 computer at the IBM New York Scientific Center in cooperation with the American Printing House for the Blind. This program was put into operation at the Printing House in 1964 using an IBM 709 system. The 704 translation program and associated programs could be run on the 709 under compatibility.

By means of these programs the computer could be used effectively to translate Braille literature for general reading. The translation program also provided a basis for experience with Braille translation as a computer task and acquaintance with problems which might arise in a broader application.

By 1966, a number of revisions had been made in the original program to improve accuracy in certain situations encountered in the translation of literary Braille. A textbook translation section had been added to provide for format features required in Braille educational texts and other special materials. Experimentation had also been done on a new approach to programming the conversion from inkprint to Braille code. There appeared to be a need for a new computer program which would unify and expand a number of concepts which in actual production or in experimentation had seemed to be valid for Braille translation.

Under this grant a new program was written and was in operation at the beginning of 1968. The program, written for the IBM 709, incorporates two general principles. One of these is that the letter sequences corresponding to the various Braille signs are used as the basic units of translation. The other principle is that a distinction is made between general rules and exceptions to the rules. In the processing of a unit, applying the general rules and the exceptions are effected by different procedures. The 1968 program continues a number of significant concepts from the original Braille translation program, especially in regard to input and output format and table structure. It also reflects the revisions and additions which had been made on the basis of experience in the translation of several hundred Braille volumes.

In the extension of program capabilities, some of the aims were to provide for somewhat easier operation, to give more flexibility, to increase accuracy of translation in some instances, and to make available added format control.

Under the new program, specifications for the Braille book such as page size and page number are supplied by the dollar sign type of codes. Normally these codes will appear at the beginning of the input text but they may be inserted within the text. For example, if two or three volumes are translated in one run, the volume number may be changed by inserting the volume number symbol before each volume.

Format type such as literary or textbook is determined from a symbol in the input. Previously this had been done by inserting cards in the program deck.

As a further aid to operation, the running head for a book is translated by the 1968 program. The earlier program had required specifying Braille characters of the running head. If desired, it may now be changed within a volume. Braille editing occasionally requires that the heading of the first page of a book be different from the heading of successive pages.

The processing of dollar sign codes containing numerical factors has been changed. Instead of matching the entire code with a table entry, only the alphabetical part of the code is matched with the table. The number is treated as a variable. This gives greater flexibility for titles, letter headings, outlines, and indexes where the number of lines or number of spaces indented must be indicated. The desired value, even if unusual, may be used without checking to see that it has been provided for in the program table.

Input to translation is treated as a succession of characters or of groups of characters rather than as spaced words. This makes it possible to translate more accurately a long series of words joined by hyphens or dashes. Such series have been encountered in several literary works.

In the 1968 program a larger number of preceding and following characters is available for examination while a unit is being translated. This should be advantageous in some instances.

In the case of numbers preceded by an apostrophe, the number sign can now be inserted properly by the program. Where it was previously necessary in a combination such as, '76, for the keypunch operator to give an indication of the number sign, the operator may now follow copy. Steps have been taken also to prevent division of fractions from associated whole numbers at the end of a line. In Braille, two numbers joined by a hyphen may be separated at the end of a line if both are whole numbers but may not be separated if the first is a whole number and the second is the numerator of a fraction.

The magazine feature of leaving a blank area for the Braille page number, when this is preferable to inserting the acutal number, has been extended to literary format. This helps make it possible to use the computer to do work on title pages which has usually been done manually.

Another feature of the 1968 program which is helpful for title pages and elsewhere is the alternate right margin. An alternate line length may be specified for lines which are not to be extended as far to the right on the page as full lines. In a table of contents, for example, description lines may be left shorter than the lines which show page location and extend to the normal right margin.

Provision has been made for indicating, by means of input symbols, insertion of colon lines or centered lines of dots in the Braille text.

A facility for setting up columned material had been included in the textbook section which had been added to the original program. In the 1968 program this facility is augmented. The number of columns which may be used across the page has been increased. It is also possible, using the new program, to specify either left or right alignment of material in a particular column. In a table containing both alphabetical and numerical information, it may be desirable to have some columns in which the left-hand characters are vertically aligned and other columns in which the right-hand characters show vertical alignment.

Approximately 9,400 words of 709 storage is required by the new program for tape to tape operation. An additional 4,200 word area is used for storing input and output data when reading and punching cards. This is a total of 13,600 words. Translation proceeds at the rate of a little more than 12 Braille pages per minute or approximately 2,300 words of text per minute.

Along with the 1968 Braille translation program a new Braille output program has also been developed. This program enables the computer to punch Braille cards from a Braille tape. It provides, too, for the printed listing of the results of translation in which Braille cells are represented either by printed dots or by two-digit octal numbers.

The listing from the output program can be made to show, beneath each Braille line, the inkprint equivalents of the Braille signs. Where a Braille character or series of characters can stand for more than one sign, the program is designed to show in the inkprint line which sign has been used. For example two L characters in Braille may represent successive signs for the letter "l" or they may together constitute the sign for "little". In one case, the inkprint line would have LL; in the other case, it would have the letters LITTLE enclosed by parentheses. As part of Braille production procedure, a test translation is often made of a group of problem words apart from the context. In this situation it is of value when checking for possible translation errors to have the signs clearly identified. For example, if a Braille word having the characters AGE occurred in translation, the accompanying inkprint line from the output program would show whether the word was made up of signs for "a", "g", and "e" (correct Braille usage), or was a combination of the sign for "again" and the sign for "e" (incorrect Braille usage).

PROOFREADING IN BRAILLE PRODUCTION

From the time when Braille production by computer was first begun, there was some question regarding the necessity for proofreading the resulting Braille text. This is really a two-pronged question related to the accuracy of the computer output and to the reliability of the automatic plate-making process. Proofreading might be avoided if the many possible translation situations could be accommodated to a sufficient degree by computer processes and if the automatic stereograph operation were consistently accurate enough in transferring data from punched cards to metal plates.

The production of a significant number of titles showed that some errors continued to appear in the embossed plates and that proofreading and correction were needed to assure high quality.

A further question involved the time in the production process at which proofreading would be most effective. Production steps include keypunching input text, translation of the text by the computer in a magnetic tape to tape operation, punching cards by the computer, and embossing plates automatically from the punched cards. The standard proofreading procedure is to read proofs of the metal plates produced by the embossing machine.

In the computer translation process it is possible to produce a printed dot listing (with inkprint sublines if desired) as a means of checking translation results. If proofreading were to be done at this point it would detect errors in computer translation. Changes could then be made before punching so that the punched cards would be correct and it would not be necessary to make corrections on plates for translation errors. However, any errors occurring in the card-to-plate operation would not be detected.

A study was made to determine the feasibility of interposing proofreading following computer translation rather than after embossing. Five books were selected to be listed following computer translation and read by a Braillist. Where errors were found corrections were made so that the errors would not appear in the punched cards. After these five books were embossed proofs from the plates were read by the standard procedure.

Results were as follows:

Book	Number of Braille Pages	Plate Errors
A	280	20
B	220	2
C	318	3
D	434	1
E	589	4

Some errors appeared on the plates which were not in the punched cards.

The text of three additional books was analyzed with respect to errors which had been discovered when reading proofs from the Braille plates. These errors were compared with contents of the computer punched cards for these books.

Book	Number of Braille Pages	Errors Found on Plates but not in Cards
F	254	12
G	427	37
H	409	16

In the eight books there were 2,931 Braille pages. Ninety-five errors were found in the plates for which there were not corresponding errors in the punched cards.

ECONOMIC FACTORS IN AUTOMATED BRAILLE PRODUCTION

At the American Printing House in 1954 a thorough cost analysis was made of Braille book production from editing through binding. At that time translation, that is, conversion of inkprint text to Braille plates, was identified as the major cost of production. This is the part of the production process which can now be accomplished by automation.

Because translation represents such a large proportion of the expense in producing literature for blind readers, the automated procedure has been analyzed for cost distribution and a comparison made with the manual procedure.

In the manual method of Braille translation an operator transcribes from inkprint text to Braille plates by means of a stereograph machine. Operators must know the Braille code and also be skilled in the use of the stereograph equipment. After the metal plates are embossed, proofs are made from them and read for errors in transcribing. Following this operation, errors which have been found are corrected in the plates. Correcting involves the use of a special tool to pound out the erroneous dots. Plates are then reinserted in the stereograph machine to complete correction.

The percentage of cost for operations in the manual procedure is:

1.	Embossing	31%
2.	Proofreading	36%
3.	Correcting	22%
4.	Other	11%

In the automated Braille translation procedure the final steps of proofreading and correcting, which follow plate-making, correspond to the later steps in the manual procedure. Otherwise the operations for automated translation are very different from the manual process.

The first step in automated Braille production is to copy literature into computer readable form. This operation consists of an operator, without a knowledge of Braille, punching cards on a keypunch machine.

Using the punched cards as input the computer converts the inkprint text into Grade 2 Braille. Translation output is in the form of punched cards containing the Braille code.

The output cards are read by automatic stereograph machines that produce a metal plate in accord with the instructions contained in the cards.

The plates are then proofread and any errors corrected.

The percentage of cost for automated Braille operations is as follows:

Operation	1964	1969
1. Keypunching and Verifying	52%	50%
2. Computer operation *	06%	10%
3. Plate making	08%	06%
4. Proofreading	24%	21%
5. Correcting	04%	03%
6. Other	06%	10%

* Cost of computer time is not included. This operation requires a skilled Braillist.

Although the computer plays a significant role in Braille translation, the above six operations require a total of 80 hours to complete 150 Braille plates.

The cost of a Braille plate produced by computer is in the same range as a manually produced plate.

ABSTRACT OF PROGRAM FOR BRAILLE TRANSLATION

In order to show the logical steps performed by a computer in Braille translation, a document was prepared entitled, "A Program Abstract for Translating Inkprint into Braille by Computer". The abstract parallels the 1968 Braille Translation Program written for the IBM 709 and represents the approach to Grade 2 Braille translation which forms the basis of that program. Terminology restricted to a particular computer system is avoided. Operations are described which could be accomplished by any digital computer having sufficient capacity for the application. The abstract is arranged so as to have the possibility of being restated in a high-level computer language or an assembler language. A copy of the abstract accompanies this report.

POTENTIAL OF IBM 360 FOR BRAILLE TRANSLATION

In view of interest in Braille translation by third generation computers, it seemed desirable to develop a benchmark program in this area for the IBM 360 series.

A problem was required which would be of the type encountered in Braille translation and which would be primarily a test of internal processing capability. The problem selected was that of locating and marking in an inkprint text all of the occurences of the letter groups which are sometimes contracted in Braille. Whether or not the contraction should be used in the specific case was not made part of the problem. Letter groups, even if overlapped, were to be indicated by parentheses.

Text example: READ CONE OTHER RECEIVE

Processed: R(EA)D (C(ON)E) O((TH)(E)R) (RECEIVE)

A program to "Mark Letter Groups for Possible Contracting" was written for the 360 in Assembler Language. Another program to solve the same problem was written for the IBM 709 system, a system now being regularly used in Braille production. Test data containing the letter groups of the 189 Braille contractions was given as input to both programs with identical results.

When the 360 program was assembled the processing part of the program exclusive of table and input/output routines required approximately 960 bytes of storage. For the 709, the processing part of the program required approximately 340 words. The input/output routines for the 360 required much more storage than input/output for the 709 in the specific systems which were used.

Running time was as follows:
Test material: 17,472 words of text from "Architecture of America"
Time on 709: 5 min. and 2 sec.
Time on 360 Model 30: 5 min. and approximately 25 sec.
Words per minute on 709: 3470
Words per minute on 360/30: 3220

Input and output was on magnetic tape and consisted of 1707 input records and 2240 output records. In the operation less than 6 records were read per second and about 7 records were written per second, so that time for reading and writing should not have been significant.

Experience with this program indicated that the 360 series was effective in solving a problem related to Braille translation. The system compared favorably in several respects with the 709 which has been used effectively in Braille translation for several years. The 360 required only about two-thirds as much storage for the processing part of the program as the 709 required. The 360 program was also assembled from fewer source statements. The closeness of the speed of the Model 30 to that of the 709 suggests that a 360 series computer of the next larger size, Model 40, could be expected to equal or exceed the speed of the 709 in translating inkprint into Grade 2 Braille.

The 709 system at the American Printing House for the Blind was, of course, used for running the 709 program. By means of simulation programs written for the 709, it was also possible to use this system to develop and make preliminary tests of the 360 program.

Two simulation programs have been developed to enable the 709 to process machine instructions used by the 360 and to effect the 360 pattern of storage. Input and output capabilities of the 360 are not directly simulated. The purpose is to test the applicability of the 360 logic and data structure to specified problems.

Under control of a "Simulate 360 Execution" program the 709 will accept most of the 360 Standard Instruction Set except input/output instructions. Features such as addressing individual bytes, use of fixed and variable length operands, complement arithmetic and register operation become available. A special instruction will supply data to storage in 8-bit format from a 709 input unit. Another special instruction will move data from storage to a 709 output unit. The system simulates storage of 96,000 bytes.

A second 709 program, "Simulate 360 Assembler", converts source statements which have been written in 360 Basic Assembler Language to 360 object code. The program is designed to provide hexadecimal instructions, hexadecimal constant equivalents, storage allocation, and base register assignment from free form source language data. Errors in operation codes or syntax are printed out by the program. Following assembly, a listing of the 360 program is provided and an object deck may be punched for use with the "Simulate 360 Execution" program.

The 360 benchmark program which was assembled and tested on the 709 using simulation was later successfully assembled and run on a 360 series machine by changing only the input and output statements.

CONTEXTS AFFECTING BRAILLE CONTRACTIONS

In Grade 2 Braille translation there are many cases in which Braille contraction usage depends upon a word's pronunciation, syllabification, or meaning. Achieving the correct code in these cases constitutes an important part of the translation task.

This aspect of translation, when processing is by computer using the 1968 Braille Translation Program, depends to a large extent upon a consideration of letter sequences. A sequence of letters, such as AR, FOR, or ING, corresponding to one of the 189 Braille contractions may be thought of as a "contraction sequence". A sequence made up of a contraction sequence plus one or more adjacent letters of a word may be defined as a "contraction context". Such contraction contexts have considerable significance for Braille translation.

As literature of various types is translated, it is inevitable that unusual words and variant spellings will be encountered along with new words coming into usage. For this reason the part of the translation program which handles contraction contexts requires periodic revision. There is also in some instances a problem in specifying contexts which are unique. Because of these difficulties, research was undertaken to obtain more information concerning contraction contexts.

Data used as a basis for this investigation was a word list of approximately 85,000 words. This included a group of about 10,000 common words based on lists from THE TEACHER'S WORD BOOK OF 30,000 WORDS.* The remaining words, approximately 75,000, were derived from writings by numerous authors. Approximately 7,500,000 words of text from 125 different titles were scanned. Each different word was noted and those words not found in the group of 10,000 common words were retained. A series of programs was written for the computer to process this data.

A search was made of the 85,000 words to locate contraction sequences and to record the contexts in which they occurred. Those cases in which contraction usage could be determined by computer processing on the basis of general rules were eliminated. For example, AS is not contracted in "ask", GG is not contracted in "egg", and the COM contraction is not used in "become" because of general rules for those contractions. However, those cases were noted in which, for computer processing, determination of the translation of the contraction sequence required an examination of the specific letters surrounding the sequence. There were 66,000 such occurrences. This data, consisting of contraction sequences in words, was sorted and arranged into a master list.

---

* Thorndike, Edward L., and Lorge, Irving, THE TEACHER'S WORD BOOK OF 30,000 WORDS, New York, Columbia University, 1944.

A program was developed to retrieve desired information from this file. When an entry, made up of a contraction sequence and specified adjacent letters, is submitted to the program, all the words in which this combination occurs will be printed out.

Further research was done on the master list of 66,000 occurrences of contraction sequences. A program was written to analyze this data in order to discover which contexts are significant for the translation of the various contractions. The program identified 15,000 contraction contexts as having determinative value. These contexts are listed in the booklet, "Key Braille Contraction Contexts". The list of key contexts can not be regarded as final but is indicative of the situations in which contraction sequences are encountered.

CONCLUSIONS

Developments were made and information obtained in important areas of computer translation of Braille.

A comparison of examples of hyphenated and unhyphenated Braille text showed that omitting hyphenation (as in computer translation) requires 0.7% more space. This difference does not appear to justify the added computer capacity and programming which would be necessary for hyphenation. This study considered only the factor of space saving. Other factors should be considered relative to hyphenation. One of these is comparative ease of reading of the two formats. Another is the possible value of reading hyphenated text as a means of learning syllabification of words. It would seem that the major effort involved in a Braille hyphenation program should not be undertaken until further research is done.

A new computer program was developed for translating inkprint into Grade 2 Braille. This 1968 Braille Translation Program provides increased flexibility and accommodates a wider range of text formats. Greater accuracy in translation has been made possible and, utilizing the capacity of the IBM 709, time required for processing has been lessened. The new program also contains a number of features which make for easier operation.

In the work which has been done on the new program it has been recognized that in putting inkprint into Grade 2 Braille, there are decisions which, at the present time, cannot be made by computers. Some human intervention is required in input preparation and monitoring of the computer processing operations. Part of the purpose of the program is to increase the effectiveness and minimize the amount of this intervention. Certain aspects of Braille translation could be facilitated by means of human interaction with computer video displays. Consideration might be given to this in future programming.

From an analysis of errors found in plates produced by the automated translation system, the need for proofreading to meet publication standards was confirmed. Results from experimentation with proofreading before and after plate-making indicated that some errors occurred in the card-to-plate process as well as from computer translation. Even though error correction during the final phase of computer processing is easier than correction of metal plates, the number of instances in the sample material where plate errors were found without corresponding card errors seems to show that proofreading done at the conclusion of translation to assure correct punched card output could not be substituted for proofreading at the time plates are completed. This study suggests that some investigation might be made of the possibility of a device for verifying data transmission in the card-to-plate process.

A study of cost distribution records indicated the major economic factors in automated Braille translation. Costs may be separated into three approximately equal divisions: input preparation, computer operation and output processes, and computer rental. All of these areas have a potential for cost reduction from technological advances in the printing, computer, and communications industries which will contribute toward the solution of the plate cost problem in Braille production.

Accompanying this report is "A Program Abstract for Translating Inkprint into Braille by Computer". This abstract documents the approach to translation used in the program now in operation at APH.

To explore the potential of third generation computers for Braille translation a test program was written for the IBM 360 system. The program provided a test of processing speed, instruction set, and storage efficiency in solving a problem typical of those encountered in translating Braille. Results demonstrated the effectiveness of the 360 series for this application. A computer having the capacity of the Model 40 appears to be the appropriate size for supporting Braille production.

An analysis was made of the different words, drawn from a considerable amount of text, which contain the letter sequences corresponding to the various Braille contractions. An attempt was made to determine which groups of adjacent letters are significant for use and non-use of the contractions. The list given in "Key Braille Contraction Contexts", which supplements this report, presents the results of a survey of contraction usage in various settings of pronunciation and syllabification.